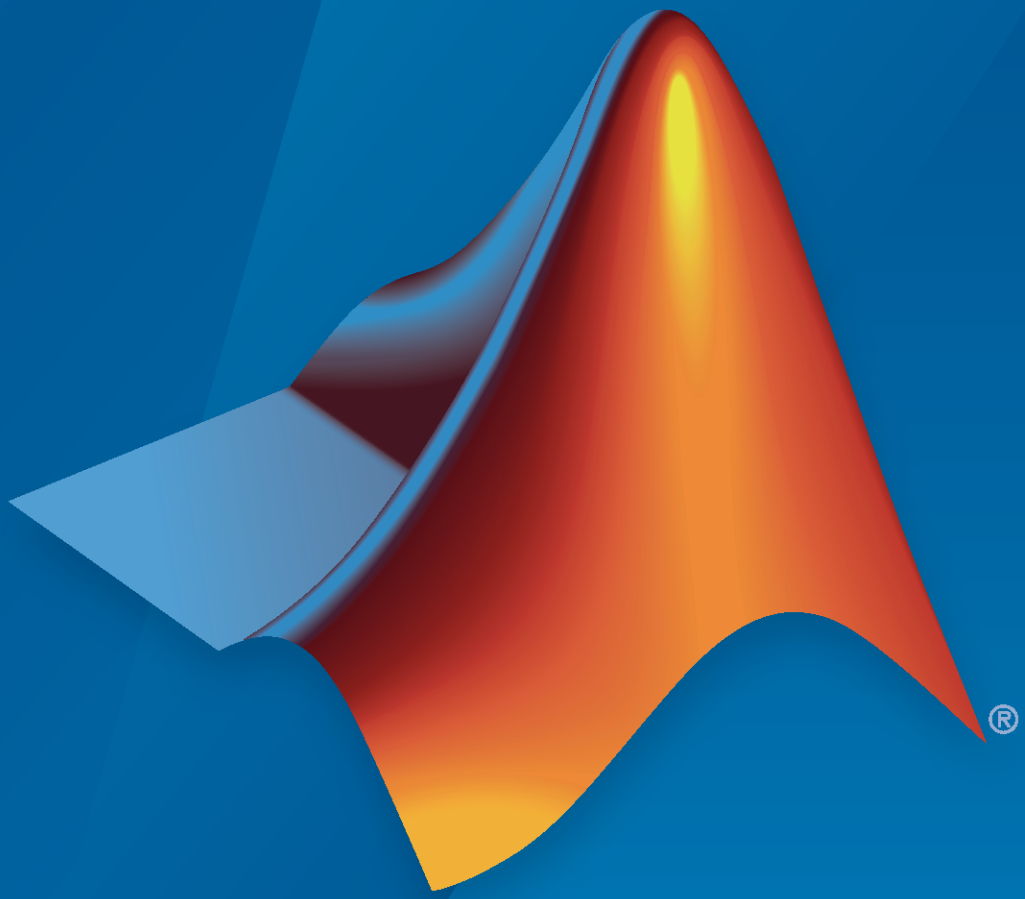
MATLAB® Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

MATLAB® Release Notes

Contents

R2020a

R2019b

R2019a

R2018b

R2018a

R2017b

R2017a

R2016b

R2016a

R2015aSP1

Bug Fixes

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

R2012b

R2010bSP2

Bug Fixes

R2010bSP1

Bug Fixes

R2010b

R2010a

R2009bSP1

Bug Fixes

R2009b

R2009a

R2008b

R2007b

R2007a

R2006a

R14SP3

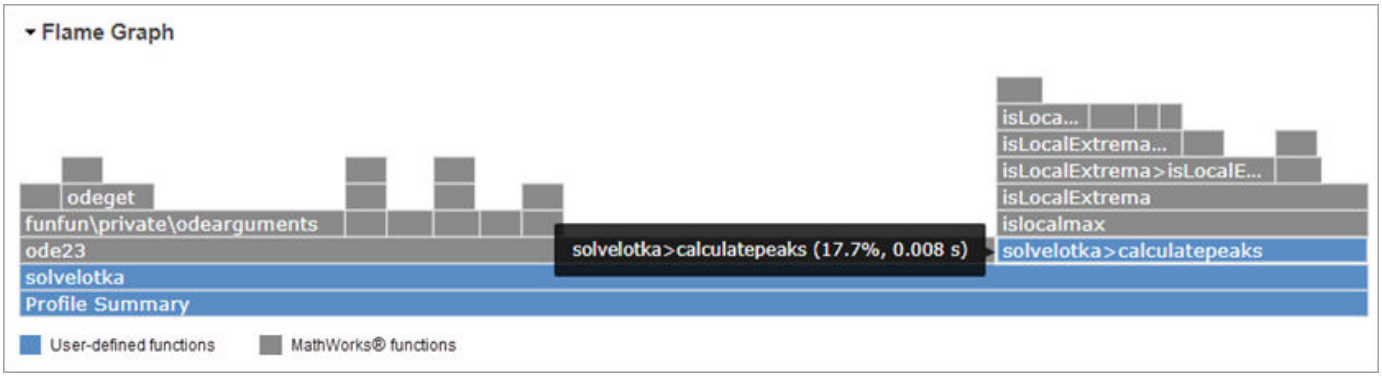
R2020a

Environment

Profiler Flame Graphs: Investigate and improve the performance of your code visually

The redesigned Profiler now includes a flame graph that allows you to visually explore the execution performance results of your code. You can use the flame graph to identify functions that use a significant amount of time.

For example, this flame graph shows the performance results for a function called `solvelotka`. The wide graphs represent the functions that use the most time. You can use the Profiler to explore those functions and determine whether execution can be sped up.



For more information, see “Profile Your Code to Improve Performance”.

Live Editor Loop Execution: Improved performance when running loops in live scripts

Loops run significantly faster in live scripts. For example, a live script containing this code, which runs a loop one million times and displays the current loop iteration every ten thousand iterations, is approximately 42x faster.

```
for t = 1:1000000
    if ~mod(t, 10000)
        disp(t)
    end
end
```

The approximate execution times are:

R2019b: 2.291 s

R2020a: 0.054 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

The more iterations a loop contains, the greater the performance improvement becomes.

Live Editor Animation Output: Improved performance when animating plots in live scripts

For-loop animations display faster in live scripts. For example, a live script containing this code, which creates a for-loop animation of a sine wave plot, is approximately 1.3x faster.

```
tic
x = 0:0.1:10*pi;
y = sin(x);
xlim([0 10*pi])
ylim([-1 1])
hold on
p = plot(x(1),y(1));
for k=1:length(x)
    p.XData = x(1:k);
    p.YData = y(1:k);
   drawnow
end
toc
```

The approximate execution times are:

R2019b: 8.875 s

R2020a: 6.633 s

The code was timed on a Windows® 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

Live Editor Responsiveness: Improved performance with extended use

The Live Editor maintains its interaction performance (such as typing and scrolling) when running MATLAB over extended periods of time. In previous releases, the interaction performance of the Live Editor decreased over time.

Live Editor Control Value Changes: Run all necessary code on value changes

You can configure a control to run the current section and any stale code above it when the control value changes. This ensures that when the value of the control changes, any modified or not yet run sections above are run as well. To configure the control, right-click the control and select **Configure Control**. Then, in the **Run** field, select the **Current section and modified or not yet run sections above** option.

For more information about adding controls to a live script, see “Add Interactive Controls to a Live Script”.

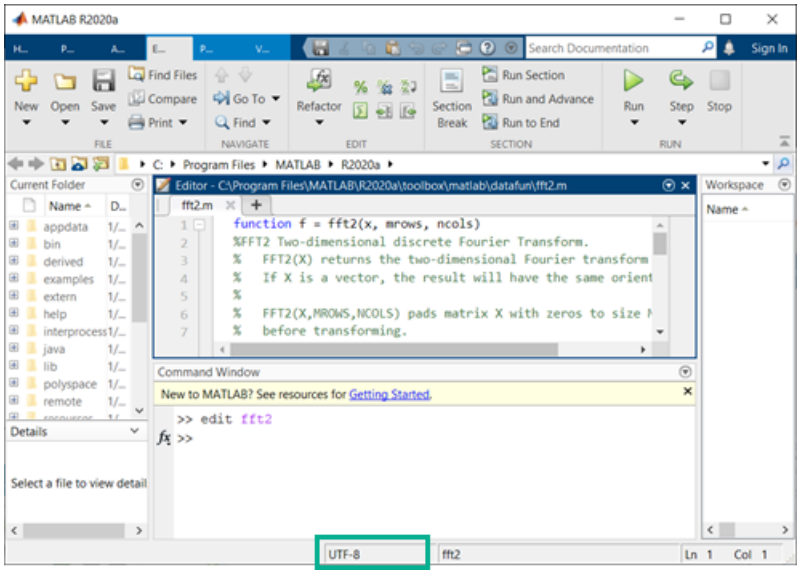
File Encoding: Save MATLAB code files (.m) and other plain text files as UTF-8 encoded files by default

As of R2020a, MATLAB defaults to saving new plain text files using UTF-8 without a byte-order-mark (BOM). This includes files created with the MATLAB Editor and the `edit` or `fopen` functions, as well

as MATLAB log files and files created with the `diary` function. In the Editor, this includes all MATLAB code files with a `.m` extension, such as scripts and functions. When opening existing files, the Editor and other functions like `type` or `fopen` automatically determine the current encoding. The Editor saves files with their current encoding unless a different one is selected from the **Save As** dialog. For example, to save a file using the legacy locale-specific encoding for compatibility with an earlier release of MATLAB, select **Save > Save As...** in the **File** section on the **Editor** tab. In the dialog box that appears, select the desired encoding from one of the **Save as type** options.

MATLAB uses Unicode® internally so that it can represent all letters and symbols, regardless of platform, language, or locale. UTF-8 was adopted as MATLAB's default character encoding to ensure that all Unicode code points can be correctly represented in files and byte streams. MATLAB also supports other character encodings for backwards compatibility and interoperability. For more information, see “Locale Setting Concepts for Internationalization”.

The current encoding is displayed next to the file name in the Editor status bar or, if the Editor Window is docked, the Desktop status bar.



Multiple Sources in Help Browser: Search MathWorks documentation and custom documentation together in a single browser

When you search the documentation in the Help browser, the Help browser displays both MathWorks® documentation results and installed custom documentation results. To switch between the two types of results, use the **Source** facets that appear on the left side of the page. For example, to view the MathWorks documentation results, select `MathWorks` as the source. To view the installed custom documentation results, select `Supplemental Software` as the source.

Web Documentation: View MathWorks documentation on the web without logging in

If your documentation preferences are set to view documentation on the web, you now can view the documentation for most products without logging in.

For more information about setting your documentation location, see “Help Preferences”.

Internationalization: UTF-8 as system encoding on Mac and Windows platforms

On the Mac platform, MATLAB uses UTF-8 as its system encoding to align with macOS.

On the Windows platform, if the **Use Unicode UTF-8 for worldwide language support** option is enabled in the Windows **Region** settings dialog box, then MATLAB uses UTF-8 as its system encoding.

Language and Programming

switch Function: Compare objects more flexibly

MATLAB enables you to use objects of a class in `switch` statements if the objects support the `eq` function. In previous releases, you can use objects of a class in `switch` statements only if the output of the overloaded `eq` function is a logical value. Starting in R2020a, the output of `eq` can be either a logical value or convertible to a logical value. For more information, see “Objects In Conditional Statements”.

copyfile and movefile Functions: Access web-based storage services like Amazon Web Services and Azure Blob Storage

You can now use the `copyfile` and `movefile` functions to work with remote files and folders. To access remote locations, you must specify the full path using a uniform resource locator (URL). For example, copy a file from Amazon S3™ Cloud to the folder `myFolder`.

```
mkdir myFolder
copyfile s3://bucketname/path_to_file/my_image.jpg myFolder
```

For more information on setting up MATLAB to access your online storage service, see “Work with Remote Data”.

dbup and dbdown Commands: Switch between workspaces with one step

When you debug your code, you can now change the current workspace and function context to any workspace and function context on the stack with one step. Specify the number of levels you want to move on the stack as an input argument to the `dbup` and `dbdown` commands.

bin2dec and hex2dec Functions: Convert text that includes binary or hexadecimal prefixes and suffixes

The `bin2dec` and `hex2dec` functions convert text inputs that include the same prefixes and suffixes used for writing binary and hexadecimal literals.

For example, these calls to `bin2dec` return the same value:

```
bin2dec('111')
bin2dec('0b111')
bin2dec('0b111s32')
```

Similarly, these calls to `hex2dec` return the same value:

```
hex2dec('FF')
hex2dec('0xFF')
hex2dec('0xFFs32')
```

For more information on the syntax for using these prefixes and suffixes, see “Hexadecimal and Binary Values”.

dec2bin and dec2hex Functions: Convert negative numbers

The `dec2bin` and `dec2hex` functions convert negative numbers using their two's complement binary values.

For example, these calls to `dec2bin` and `dec2hex` convert negative numbers.

```
dec2bin(-1)

ans =

    '11111111'

dec2bin(-16)

ans =

    '11110000'

dec2hex(-1)

ans =

    'FF'

dec2hex(-16)

ans =

    'F0'
```

complex Function: Create sparse complex arrays

You can now create sparse complex arrays from sparse input arguments using the `complex` function.

Enumeration classes: Hide member names for compatible name changes

The enumeration block `Hidden` attribute enables enumeration class authors to hide member names from class users. Hiding enumeration members enables you to replace existing names with new names without introducing code incompatibilities. For more information, see “Hide Enumeration Members”.

matlab.mixin.SetGet: Set priority for partial property name matching

Classes that derive from `matlab.mixin.SetGet` can use the `PartialMatchPriority` property attribute to specify a relative priority for partial name matching. MATLAB applies this attribute when resolving incomplete and case-insensitive text strings that match more than one property name. For more information, see “Set Priority for Matching Partial Property Names”.

Class logical conversions: Support logical conversion more flexibly when writing classes

When MATLAB requires a logical value for expressions like `if` and `while` statements, it attempts a direct conversion using the `logical` converter function. If the result of the conversion is a nonlogical

value, then MATLAB attempts to call the `cast` function with the `"like"` flag. If the class of the object being converted defines a `cast` method that supports the `"like"` flag, and this method returns a logical value, then MATLAB uses this logical value. Otherwise, MATLAB throws a `MATLAB:invalidConversion` error.

Functionality being removed or changed

File Operations: Wildcard expression *.* on UNIX platforms matches only files that have an extension

Behavior change

Starting in R2020a, on UNIX® platforms, the wildcard expression `*.*` no longer matches folders or files without an extension. In previous releases, the expression matches folders or files regardless of extension, including files without an extension. This change of behavior does not apply to Microsoft® Windows platforms. This change affects the functions `copyfile`, `delete`, `dir`, `movefile`, and `rmdir`.

copyfile Function: Symbolic links are treated consistently on platforms and file systems

Behavior change

Symbolic links (or symlinks) are file system objects that point to target files or folders. Starting in R2020a, the behavior of the `copyfile` function changes when operating on symlink files or folders.

- **Copy outcome is platform independent:** `copyfile` now treats symlinks on different operating systems in the same way. For example, consider a folder structure with a file `myFile.m` and a symlink pointing to `myFile.m`, specified as `symlinkToMyFile`.

  ```
  myFile.m
  symlinkToMyFile
  ```

  Starting in R2020a, `copyfile('symlinkToMyFile','newFile')` copies the target of the symlink (that is, `myFile.m`) to the destination. In previous releases, on Linux®, `copyfile` copies the symlink instead.

Platform	newFile (Starting in R2020a)	newFile (R2019b and Earlier)
Linux	`myFile.m`	`symlinkToMyFile`
Mac	`myFile.m`	`myFile.m`
Windows	`myFile.m`	`myFile.m`

- **`copyfile` copies only the contents of the source folder:** When copying a nonempty folder to a symlink folder, `copyfile` now copies the contents of the source folder (that is, the files and folders within the source folder) rather than the entire source folder. Similarly, when copying a symlink folder to a destination folder, only the contents of the symlink folder are copied.

  For example, consider a folder structure on Linux consisting of a nonempty folder `myFolder` and a symlink to that folder named `symlinkToMyFolder`. This table shows the folder structure after the execution of `copyfile('myFolder','simlinkToMyFolder')` in different MATLAB releases.

Folder Structure Before Copy	Folder Structure After Copy (Starting in R2020a)	Folder Structure After Copy (R2019b and Earlier)

`myFolder`
`    myFile.m`
`symlinkToMyFolder`
	`myFolder`
	`    myFile.m`
	`symlinkToMyFolder`
	`    myFile.m`
		`myFolder`
		`    myFile.m`
		`symlinkToMyFolder`
		`    myFolder`
		`        myFile.m`

Toolbox folders renamed: Old names will be removed
Still runs

Various toolbox folders have been renamed. The old toolbox folder names passed to the `toolboxdir`, `ver`, and `verLessThan` functions will be removed in a future release. Use the new names instead.

Toolbox	Old Folder Name	New Folder Name
Parallel Computing Toolbox™	`distcomp`	`parallel`
Fixed-Point Designer™	`fixpoint`	`fixedpoint`
Simulink® Real-Time™	`xpc`	`slrt`
Simscape™ Electrical™	`powersys`	`sps`

System object authoring: StringSet will be removed
Still runs

The class `matlab.system.StringSet` will be removed in a future release to bring System object™ authoring closer to MATLAB classes.

System object syntax being removed	Migration
Enumerated properties with `matlab.system.StringSet`	Replace StringSets with enumerations or property validators. See “Limit Property Values to Finite List”.

System object authoring: Logical and Positive Integer property attributes will be removed
Still runs

The System object property attributes `Logical` and `Positive Integer` will be removed in a future release to bring System object authoring closer to MATLAB classes.

System object syntax being removed	Migration
Property attributes `Logical` and `Positive Integer`	Replace these property attributes with property validators. See “Validate Property and Input Values”.

System object authoring: Several matlab.system.mixin.* classes will be removed
Still runs

These mixin classes will be removed in a future release:

- `matlab.system.mixin.CustomIcon`
- `matlab.system.mixin.Nondirect`
- `matlab.system.mixin.Propagates`

- `matlab.system.mixin.SampleTime`

To simplify System object authoring, the functionality and methods from these classes are directly included with the base System object class `matlab.System`.

System object syntax being removed	Migration
• `matlab.system.mixin.CustomIcon` • `matlab.system.mixin.Nondirect` • `matlab.system.mixin.Propagates` • `matlab.system.mixin.SampleTime`	Remove any inheritance statements to these classes from the beginning of your System object class.

ismethod Function: String and character vector in first input argument will be treated as object
Behavior change in future release

In a future release, the `ismethod` function will treat a string or character vector in the first input argument as a `string` or `char` object, instead of as the name of a class. To list the methods of a class by referring to the class by name, use the `methods` function. To determine if a specific method name is the name of a method of a class, use an expression like this:

```
any("methodName" == string(methods("ClassName"))
```

Constant properties: Defining a set or get access method for a constant property causes an error

In previous releases, creating a set or get access method for a class property defined with the `Constant` attribute resulted in a warning. MATLAB ignored these methods. Starting in R2020a, MATLAB throws an error if a class defines access methods for `Constant` properties.

Constant properties: Specifying the Dependent attribute for a Constant property causes an error

In previous releases, specifying the `Dependent` attribute for a class property defined with the `Constant` attribute resulted in a warning. Starting in R2020a, MATLAB throws an error if a class defines properties using both the `Constant` and `Dependent` attributes.

Property attributes: Specifying the Static attribute for a property causes an error

In previous releases, specifying the `Static` attribute for a class property resulted in a warning. Starting in R2020a, MATLAB throws an error if a class defines properties using the `Static` attribute.

Property and method attributes: Specifying the Visible attribute for a property or method causes an error

In previous releases, specifying the `Visible` attribute for a class property or method resulted in a warning. Starting in R2020a, MATLAB throws an error if a class defines properties or methods using the `Visible` attribute.

Defining classes and packages: Using schema.m will not be supported in a future release
Still runs

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

empty static method: Inputs must be numeric or logical
Behavior change

In previous releases, the `empty` static method accepted inputs that are convertible to numeric or logical values. Starting in R2020a, MATLAB throws an error if the inputs are not numeric, logical, or values derived from numeric or logical classes.

For example, in releases prior to R2020a, passing the scalar `char 'b'` to `empty` produces a result based on the Unicode numeric equivalent for the character b (which is the number 98).

```
a = double.empty(0,'b')
```

```
a =

  0×98 empty double matrix
```

Starting in R2020a, the `empty` method does not accept inputs that are not numeric or logical.

```
a = double.empty(0,'b')
```

```
Error using double.empty
Value must be numeric or logical.
```

Calling superclass constructor: Stricter syntax is enforced
Behavior change

Subclass calls to superclass constructors cannot be part of other expressions, be contained in conditional statements, or be made after references to the object. For more information on these restrictions, see “Subclass Constructors”.

In previous releases, MATLAB did not identify certain syntaxes as calls to superclass constructors, and therefore, did not throw errors caused by calling superclass constructors incorrectly. Starting in R2020a, MATLAB more strictly enforces the correct syntax.

The following class code illustrates some of the cases where previous releases do not identify calls to the superclass constructor because of the incorrect use of parenthesis or brackets.

```
classdef MyClass < A & B
    methods
        function obj = MyClass
            obj = [obj@A] % Not idendified as superclass constructor call in previous releases
            obj@B
            (obj@B) % Not identified as a duplicate call to B constructor in previous releases
            if 1
                (obj@B) % Not identified as a conditional call in previous releases
            end
            (obj@A) % Not identified as called after object referenced in previous releases
        end
    end
end
```

Data Analysis

Live Editor Tasks: Interactively manipulate tables and timetables, and generate code

Use Live Editor tasks to stack, unstack, or synchronize tables and timetables. Interactively explore the effects of your changes in output tables and timetables. The tasks also automatically generate code that becomes part of your live script.

In R2020a, MATLAB offers four tasks for manipulating data in tables and timetables:

- **Retime Timetable** — Resample or aggregate timetable data.
- **Stack Table Variables** — Combine values from multiple table variables into one table variable.
- **Synchronize Timetables** — Retime and combine timetables to new time vector.
- **Unstack Table Variables** — Distribute values from one table variable to multiple table variables.

To open tasks in the Live Editor, use the **Task** menu on the **Live Editor** tab. For more information, see “Add Interactive Tasks to a Live Script”.

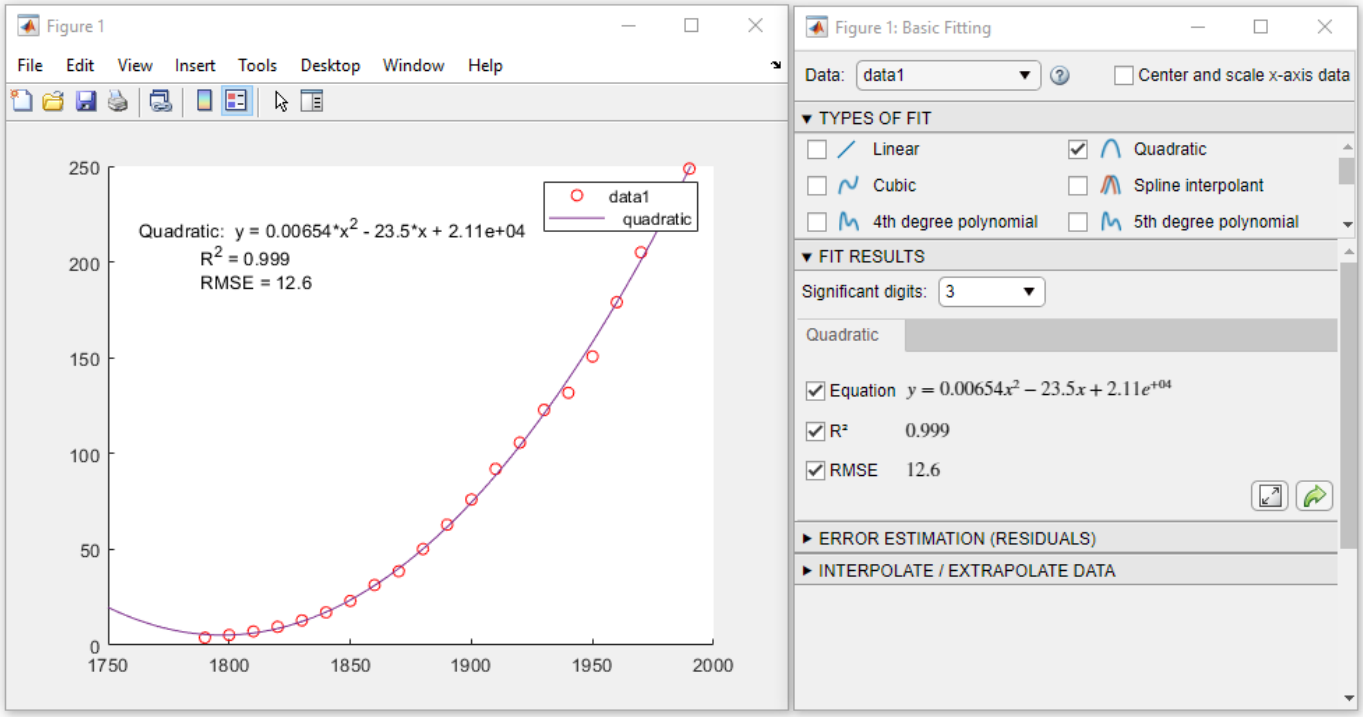
Basic Fitting Tool: Fit lines to plotted data using modernized interface

Fit lines to plotted data using the modernized interface of the **Basic Fitting** tool. Open the tool by selecting **Tools > Basic Fitting** from the figure toolbar.

For example, plot sample data.

```
load census
scatter(cdate,pop,'r')
```

Open the **Basic Fitting** tool. Select a fit and display the equation, R^2 value, and root mean square error (RMSE) value on the plot.



detrend Function: Ignore NaN values

You can now ignore `NaN` values when computing trends with the `detrend` function using the `'omitnan'` parameter.

accumarray Function: Maintain consistent output order on all platforms

The `accumarray` function now returns results in the same order as they appear in the input on all platforms.

Previously, if you used a function with `accumarray` that depended on the data order, such as `@(x) x(1)`, different platforms sometimes returned different results.

leapseconds Function: List all leap seconds used by the datetime data type

To list all the leap seconds supported by the `datetime` data type, use the `leapseconds` function. The output table that lists the leap seconds includes the dates on which they occurred, their signs, and the cumulative adjustments. To determine the International Earth Rotation Service (IERS) Bulletin C version number of the leap second data being used in MATLAB, also use the second output argument. For more information, see IERS Bulletins.

To list leap seconds, use either of the following syntaxes:

```
T = leapseconds
[T,vers] = leapseconds
```

timezones Function: Determine IANA Time Zone Database version

To determine the Internet Assigned Numbers Authority (IANA) Time Zone Database version supported by the `datetime` data type, use the second output argument of the `timezones` function.

```
[T,vers] = timezones
```

For more information on the IANA Time Zone Database, see IANA Time Zone Database.

renamevars Function: Rename variables in table or timetable

You can rename variables in a table or timetable using the `renamevars` function.

rows2vars and unstack Function: Use naming rule to allow table and timetable variable names with any characters

Starting in R2020a, you can specify the rule for naming table and timetable variables when you use the `rows2vars` or `unstack` functions. Specify the rule using the `'VariableNamingRule'` name-value pair argument.

Value of `'VariableNamingRule'`	Rule
`'modify'` (default)	Modify variable names to be valid MATLAB identifiers
`'preserve'`	Preserve original names that can have any Unicode characters, including spaces and non-ASCII characters. **Note:** In some cases, the function must modify original names even when `'preserve'` is the rule. Such cases include: • Duplicate names • Names that conflict with table dimension names • Names that conflict with a reserved name. • Names whose lengths exceed the value of `namelengthmax`.

Previously, these functions modified table and timetable variable names when necessary so that such names were always valid MATLAB identifiers.

containsrange, overlapsrange, and withinrange Functions: Determine if timetable row times intersect specified time range

To determine if the row times of a timetable intersect a specified time range, use one of these functions.

Function	Purpose
`containsrange`	Determine if timetable row times contain specified time range
`overlapsrange`	Determine if timetable row times overlap specified time range
`withinrange`	Determine if timetable row times are within specified time range

tall Arrays: Operate on tall arrays with more functions, including groupfilter and matches

The functions listed here now support tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

- `groupfilter`
- `matches`
- `renamevars`

In addition, some functions have updated or removed limitations with tall arrays.

Function	Changes
`ismember`	`ismember` now allows both inputs to be tall arrays, as long as one of the inputs is the result of a reduction operation along the first dimension (such as `sum` or `mean`). Previously, only one input could be a tall array.
`vartype`	Many tall data analysis functions can now use `vartype` to select data and grouping variables: • Preprocessing functions with a `'DataVariables'` name-value pair (such as `fillmissing`, `ischange`, and `smoothdata`) can use `vartype` to select the data variables. • Grouping functions with `groupvars` and `datavars` arguments (such as `grouptransform` and `groupsummary`) can use `vartype` to select the data and grouping variables. Previously, these functions could not use `vartype` to select data or grouping variables.

Function	Changes
`groupsummary` and `grouptransform`	`groupsummary` and `grouptransform` can now operate on multidimensional variables in a tall table or tall timetable. Previously, these functions could only operate on column vector variables.

Functionality Being Removed or Changed

The datetime function 'InputFormat' month formats M and MM do not recognize names, and MMM does not recognize abbreviations
Behavior change

Starting in R2020a, when you specify month formats in the `'InputFormat'` name-value pair argument of the `datetime` function, the `M` and `MM` formats do not recognize month names. Also, the `MMM` format recognizes only abbreviated names. In previous releases, all these formats recognized both abbreviated and full month names.

For more information on formats, see the `Format` property of the `datetime` function.

Default aggregation function for nonnumeric data in unstack
Behavior change

In R2020a, if you do not specify the `'AggregationFunction'` name-value pair argument of the `unstack` function, then the default aggregation function for nonnumeric data is the `unique` function. In previous releases, there was no default aggregation function for nonnumeric data, so `unstack` would raise an error.

Behavior changes when the aggregation function has no data to aggregate in unstack
Behavior change

In R2020a, there are behavior changes when the aggregation function specified by the `'AggregationFunction'` name-value pair argument of the `unstack` function has no data to aggregate. This situation can occur when there are no data values that correspond to values in the indicator variable after unstacking. In such cases, `unstack` essentially calls the aggregation function on an empty array.

For more information on this change in behavior, see the Compatibility Considerations section of `unstack`.

Data Import and Export

Datastores: Write data from datastore to files using writeall

You can write data from a datastore to files on disk using the `writeall` function.

Additionally, to add `writeall` functionality to custom datastores, you can use the new classes `matlab.io.datastore.FileWritable` and `matlab.io.datastore.FoldersPropertyProvider`. For more information, see “Add Support for Writing Data”.

Datastores: Return timetables from tabularTextDatastore and spreadsheetDatastore objects

`tabularTextDatastore` and `spreadsheetDatastore` objects have two new properties that enable you to work with timetables: `OutputType` and `RowTimes`. These properties specify whether the `read`, `readall`, and `preview` methods return tables or timetables.

Previously, these properties were only available in Parquet datastores.

Datastores: Partition and shuffle TransformedDatastore and CombinedDatastore objects

You can now partition and shuffle arbitrarily nested transformations and combinations of datastores, subject to these conditions:

- You can partition and shuffle a `TransformedDatastore` object only if all of its underlying datastores can be partitioned and shuffled.
- You can partition and shuffle a `CombinedDatastore` object only if `subset` can be applied to all of its underlying datastores. The underlying datastores can also be transformations or combinations of datastores that can have `subset` applied to them.

Use `isPartitionable` and `isShuffleable` to test whether a `CombinedDatastore` object or a `TransformedDatastore` object can be partitioned or shuffled, and to determine when certain combinations of datastores are fit for parallel processing. `isPartitionable` and `isShuffleable` return `true` when the underlying datastores can be partitioned or shuffled, respectively.

Datastores: Process files and blocks within files iteratively using FileSet and BlockedFileSet objects

You can process a large collection of files when moving through the files iteratively using the `matlab.io.datastore.FileSet` object. Similarly, you can process a large collection of blocks within files iteratively using the `matlab.io.datastore.BlockedFileSet` object.

Parquet Files: Control encoding scheme and Parquet version when writing files

The `parquetwrite`, `parquetinfo`, and `write` functions have two new name-value pairs:

- `'VariableEncoding'` controls whether a Parquet file uses plain or dictionary encoding for each variable.
- `'Version'` specifies whether to use Parquet 1.0 or Parquet 2.0 file formatting.

Text and Spreadsheet Files: Append, overwrite, or replace data using 'WriteMode' parameter

You can choose to append, overwrite, or replace data when writing to text and spreadsheet files by using the `WriteMode` parameter with these functions:

- `writetable`
- `writetimetable`
- `writematrix`
- `writecell`

readtable Function: Uses results of detectImportOptions function by default

Starting in R2020a, the `readtable` function uses the results of the `detectImportOptions` function to import tabular data. In essence, these two `readtable` function calls behave identically.

```
T = readtable(filename)
T = readtable(filename,detectImportOptions(filename))
```

Compatibility Considerations

There are several differences between the default behavior of `readtable` and its default behavior in previous releases. To call `readtable` with the default behavior it had up to R2019b, use the `'Format','auto'` name-value pair argument.

```
T = readtable(filename,'Format','auto')
```

The table lists significant differences between the default behavior of `readtable` in R2020a and its default behavior in previous releases.

Description of Input Fields or Rows	Default R2020a `readtable` Behavior	Default Behavior in Previous Releases
First row does not have text to assign as names of output table variables	Assigns the names `Var1,...,VarN` as the names of output table variables	Converts the values in the first row of data values to the names of output table variables
Multiple rows of text as header lines	• Ignore additional header lines • Import values in remaining rows as detected data types	• Import additional header lines as text in the first rows of output table • Import values in remaining rows as text
Empty fields	Treat as missing values for detected data type	Treat as empty character vectors or strings
Values in quotes	Treat as detected data type	Treat as text

Description of Input Fields or Rows	Default R2020a `readtable` Behavior	Default Behavior in Previous Releases
Text that cannot be converted	Treat as missing values for detected data type	Treat as text
Nonnumeric character trails numeric character without delimiter between them	Treat characters as nonnumeric	Treat numeric and nonnumeric characters as though delimiter separated them
Input text file has lines with different number of delimiters	Returns output table with extra variables	Raises error message

textscan, readtable, detectImportOptions, and setvaropts Functions: Read and import hexadecimal and binary literals

Hexadecimal and binary literals are now supported in the functions `textscan`, `readtable`, `detectImportOptions`, and `setvaropts`. This list outlines the added capabilities of each function.

- `textscan` — Use the format specifier `'%x'` to read in data stored as hexadecimal data, and use `'%b'` to read in data stored as binary data. The default data type is `uint64`. For example, `res = textscan('110101', '%b')` will return `res = 53` with the default data type `uint64`.
- `readtable` — Text that is prefixed with the characters `'0x'` is now treated as hexadecimal data and text with the prefix `'0b'` is treated as binary data.
- `detectImportOptions` — Use the name-value pairs `'HexType'` or `'BinaryType'` to convert the data from hexadecimal or binary to decimal, then set the data type of the output data. For example, `'HexType', 'uint8'` converts prefixed hexadecimal data to decimal, then sets the data type of the output to an 8-bit unsigned integer during import.
- `setvaropts` — Set the data type and number system to be used when importing variables by using these name-value pairs:
  - `'Type'` sets the data type of the resulting output variable. For example, `'Type','uint32'` sets the data type of the output variable to a 32-bit unsigned integer.
  - `'NumberSystem'` converts the number system of the input variable from hexadecimal or binary to the decimal number system. For example, `'NumberSystem','hex'` converts data that is stored as hexadecimal to decimal data. If the number system of the input variable is specified as `'decimal'`, then no conversion is applied. `NumberSystem` is also a new property of the `NumericVariableImportOptions` object.

h5read and h5readatt: Read non-scalar string data as MATLAB string arrays

The high-level HDF5 functions `h5read` and `h5readatt` now return HDF5 string arrays as MATLAB string arrays rather than cell arrays of character vectors. Single (scalar) HDF5 strings are still returned as MATLAB character vectors.

h5create and h5write: Write string data to HDF5 files

You can now write string data to HDF5 files using `h5create` and `h5write` instead of using low-level HDF5 functions. String data can be specified as MATLAB character vectors or MATLAB string arrays.

CDF Library: Upgraded to v3.7.0

The CDF library has been upgraded to version 3.7.0.

Tiff Object: Read and write the values of the Rational Polynomial Coefficients tag

You can now read and write the values of the Rational Polynomial Coefficient (RPC) tag using the `RPCCoefficientTag` tag for the `Tiff` object. For more information, see Table 6 in “Exporting to Images”.

jsonencode: Customize encoding in MATLAB classes

You can overload the `jsonencode` function to customize the JSON encoding for a user-defined MATLAB class. For an example, see “Customize JSON Encoding for MATLAB Classes”.

jsonencode: Encode enumerations

`jsonencode` encodes enumerations as strings. For example,

```
on = matlab.lang.OnOffSwitchState.on;
jsonencode(on)
```

```
ans =

    '"on"'
```

Functionality being removed or changed

readtable, writetable, textscan, and similar functions use automatic character set detection and UTF-8 encoding by default
Behavior change

As of R2020a, most functions that read text data use automatic character set detection to detect the character encoding. Functions that use automatic character set detection include `fileread`, `textscan`, `readvars`, `readtable`, `readcell`, `readmatrix`, and `readtimetable`.

Similarly, most functions that write text data use UTF-8 as the default character encoding. Using UTF-8 provides interoperability between all platforms and locales without data loss or corruption. Functions that use UTF-8 encoding by default include `writematrix`, `writetable`, `writecell`, and `writetimetable`.

File I/O functions, such as fscanf and fprintf, use automatic character set detection and UTF-8 encoding by default
Behavior change

As of R2020a, character-oriented file I/O functions such as `fscanf`, `fgets`, and `fgetl` trigger automatic character set detection when reading a file that was opened using `fopen` without a specified encoding.

Similarly, `fprintf` defaults to using UTF-8 encoding when writing a file that was opened using `fopen` without a specified encoding.

h5write and h5writeatt use UTF-8 character encoding by default
Behavior change

UTF-8 is now the default character encoding for the high-level HDF5 functions `h5write` and `h5writeatt` to ensure that all Unicode code points can be correctly represented in HDF5 files.

h5read and h5readatt return non-scalar string data as MATLAB strings
Behavior change

The high-level HDF5 functions `h5read` and `h5readatt` now return HDF5 string arrays as MATLAB string arrays rather than cell arrays of character vectors. Single (scalar) HDF5 strings are still returned as MATLAB character vectors.

web function does not return a handle or URL for pages that open your system browser
Behavior change

The `web` function does not return a handle or URL for pages that open in the system browser. This includes all external pages, which by default open in your system browser, unless configured otherwise in the MATLAB Web Preferences.

To update your code, remove the handle and URL output arguments from instances of the `web` function. This table shows examples of how you can update your code.

Before	After
`[stat,h] = web('https://www.mathworks.com','-browser')`	`stat = web('https://www.mathworks.com','-browser')`
`[stat,h,url] = web('https://www.mathworks.com','-browser')`	`stat = web('https://www.mathworks.com','-browser')`

hdftool has been removed
Errors

`hdftool` has been removed. To programmatically import HDF4 or HDF-EOS files, use the `hdfread` function instead.

Importing HDF5 files using the Import Tool is no longer supported
Behavior change

The Import Tool no longer supports importing HDF5 files. To programmatically import HDF4 or HDF-EOS files, use the `hdfread` function instead.

Mathematics

nufft and nufftn Functions: Compute nonuniform fast Fourier transforms

To compute 1-D or N-D fast Fourier transforms with nonuniform sampling, use the `nufft` and `nufftn` functions, respectively.

sparse Function: Support for integer subscripts and logical aggregation

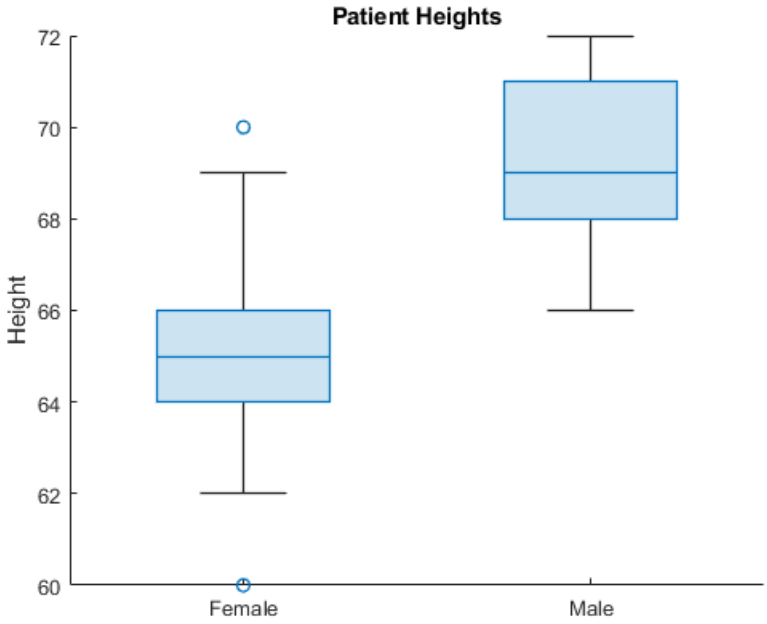
The `sparse` function has two new capabilities:

- When you construct a sparse matrix using the syntax `sparse(i,j,v)`, the subscript inputs `i` and `j` can now be integer data types.
- When the third input of the syntax `sparse(i,j,v)` contains logical values and there are repeated subscripts in `i` and `j`, the `sparse` function now applies a logical `any` operation to the values with repeated subscripts.

Graphics

boxchart Function: Visualize grouped numeric data by using box charts

To create box charts, also called box plots, use the `boxchart` function. For each group of data, the corresponding box chart displays the following information: the median, first and third quartiles, outliers (computed using the interquartile range), and nonoutlier minimum and maximum values. When you use vector data, you can use the first input argument `xgroupdata` to split your data into groups and specify the positions of the corresponding boxes. When you use matrix data, `boxchart` creates a separate box for each column in the matrix.



exportgraphics and copygraphics Functions: Save and copy graphics with improved support for publishing workflows

Use the `exportgraphics` function to save the contents of any axes, figure, chart that can be a child of a figure, tiled chart layout, or container such as a panel. This function provides a better alternative to the `print` and `saveas` functions when you want to:

- Save graphics displayed in an app or in MATLAB Online™.
- Minimize the white space around the content.
- Save a PDF fragment with embeddable fonts.
- Save a subset of the content in a figure.
- Control the background color.

The `copygraphics` function provides much of the same functionality as the `exportgraphics` function, except that it copies the content to your system clipboard instead of saving it to a file. Use this function to copy and paste content from MATLAB into other applications.

ChartContainer Class: Develop charts that display a tiling of Cartesian, polar, or geographic plots

Charts you develop as a subclass of `matlab.graphics.chartcontainer.ChartContainer` now provide a `TiledChartLayout` object, which you can use to arrange one or more Cartesian axes, polar axes, or geographic axes in your chart.

To access the `TiledChartLayout` object, call the `getLayout` method. To place one or more axes objects into the layout, call the `axes`, `polaraxes`, or `geoaxes` function, and specify the `TiledChartLayout` object as the first input argument. For more information, see “Develop Charts With Polar Axes, Geographic Axes, or Multiple Axes”.

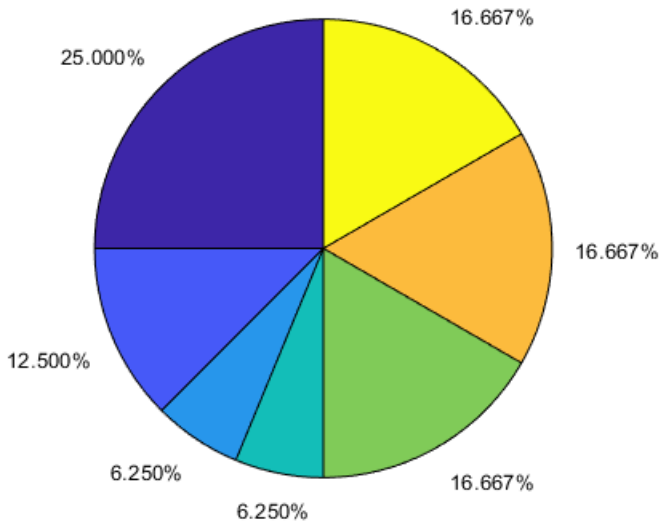
Tiled Chart Layout: Position, nest, and change the grid size of layouts

Customize layouts you create with the `tiledlayout` function by setting these properties on the `TiledChartLayout` object:

- `Position`, `InnerPositon`, `OuterPosition`, and `PositionConstraint` — For changing the size and location of the layout.
- `GridSize` — For changing the number of tiles along the rows and columns of the layout. You can set this property only when the layout is empty.
- `Layout` — For configuring nested layouts. Nested layouts consist of at least two `TiledChartLayout` objects, where the `Parent` property of one object is the other object.

pie Function: Specify a numeric format for the percentage labels

Specify the numeric format for the percentage labels on a pie chart. For example, you can specify the number of decimal places or significant digits to display.



Axes Convenience Functions: Pass an array of axes or chart objects to convenience functions such as grid, hold, and box

Modify multiple axes or charts at one time by passing a vector of objects to the `grid`, `hold`, `box`, `xlabel`, `ylabel`, `zlabel`, and `title` functions.

SeriesIndex and NextSeriesIndex Properties: Control how plots cycle through colors and line styles

Set the `SeriesIndex` property on plot objects such as `Line`, `Scatter`, and `Bar` to control how the objects vary in color and possibly line style. By default, the `SeriesIndex` for an object is a number that corresponds to the object's order of creation. MATLAB uses the number to calculate indices into the `ColorOrder` and `LineStyleOrder` properties of the axes. Changing the value of this property is useful when you want to reassign the colors or line styles of the objects in the axes.

The `NextSeriesIndex` property on the axes maintains a count of the objects that have a `SeriesIndex` property. MATLAB uses it to assign the value of the `SeriesIndex` property for each new object in the axes. The count starts at `1` and increments for each additional object. The `NextSeriesIndex` property is useful when you want to track how the objects cycle through the colors and line styles.

For more information about controlling colors and line styles, see “Control Colors, Line Styles, and Markers in Plots”.

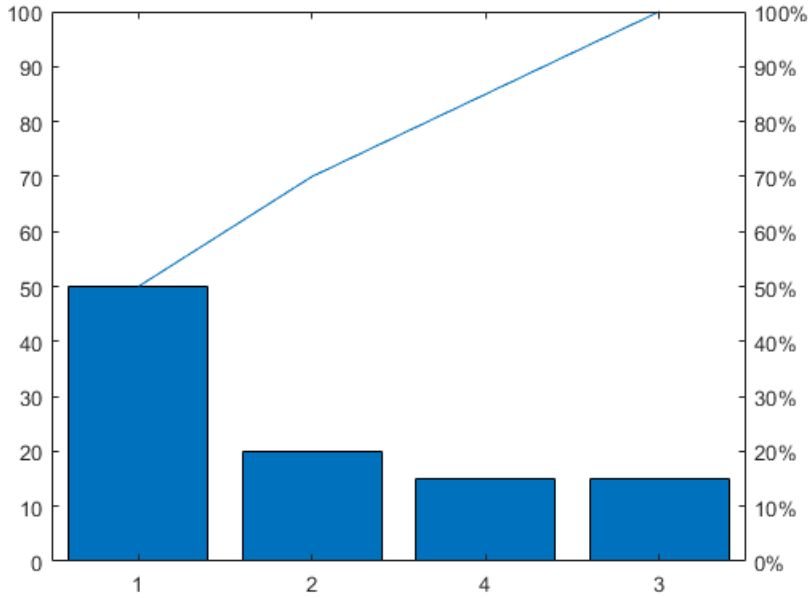
colororder Function: Control colors in scatter histograms and parallel plots

The `colororder` function now supports charts created with the `scatterhistogram` and `parallelplot` functions.

pareto Function: Specify the fraction of the cumulative histogram to include

Specify the fraction of the cumulative histogram to display in a Pareto chart as the last argument to the `pareto` function. For example, this Pareto chart includes all the bars that make up 100% of the cumulative histogram of `Y`.

```
Y = [50 20 15 15];
pareto(Y,1)
```



Axes: Control margins for titles and labels by setting the InnerPosition and PositionContraint properties

Set the `InnerPositon` property on any type of axes object to control the size and location of the plot box.

Set the `PositionConstraint` property of an axes object or a chart that can be a child of a figure to control the space around the plot box when you add or modify decorations such as titles and axis labels. This property is similar to the existing `ActivePositionProperty`. However, unlike `ActivePositionProperty`, the `PositionConstraint` property accepts the values `'outerposition'` and `'innerposition'` instead of `'outerpositon'` and `'position'`.

Built-In Axes Interactions: Explore data with cursors that show available interactions

When you hover within a set of axes, the cursor changes to indicate when you can rotate axes, create data tips, and pan axis rulers. For example, when you hover over ruler labels in 2-D axes, the cursor indicates that you can drag to pan the axis ruler by changing to  . When you hover over plots that support data tips, the cursor indicates that you can click to create a data tip by changing to  . You can always drag to pan within 2-D axes and scroll to zoom, even when the cursor indicates another interaction.

You can disable these cursor changes by setting the `Pointer` property of the figure.

```
scatter(1:10,1:10)
f = gcf;
f.Pointer = 'arrow';
```

Built-In Axes Interactions: Customize built-in interactions on geographic axes

By default, geographic axes include built-in pan, zoom, and data tip interactions. Enable and disable the default built-in interactions for `GeographicAxes` objects using the `enableDefaultInteractivity` and `disableDefaultInteractivity` functions.

You can create a customized set of built-in interactions by setting the `Interactions` property of a `GeographicAxes` object. Specify the `Interactions` property as an array of `ZoomInteraction`, `PanInteraction`, or `DataTipInteraction` objects. For more information, see “Control Chart Interactivity”.

linkdata Function: Open dialog box to specify data sources using new syntax

The `linkdata` function has a new syntax, `linkdata showdialog`, that opens the **Linked Plot Data Sources** dialog box. Use the dialog box to interactively specify data sources for plotted data.

Alternatively, instead of calling `linkdata showdialog`, you can open the dialog box by clicking **Link/Unlink Plot**  in the figure toolbar.

Functionality being removed or changed

Most properties that accept the values 'on' or 'off' now return an OnOffSwitchState value
Behavior change

Most graphics object properties that accept the values `'on'` or `'off'` now accept and return a `matlab.lang.OnOffSwitchState` value. For example, you can specify the `Visible` property of an axes object as `'on'`, `'off'`, `1`, `0`, or a logical value. A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of the property as a logical value in a conditional statement.

Any code that sets a property to `'on'` or `'off'` behaves the same way as in previous releases. However, you might need to update code that checks the value of the property. This table describes the most common situations.

Coding Pattern	Example of Coding Pattern	Updated Code
Checking for a specific character	`obj.Visible(2) == 'n'`	`obj.Visible == 1`

Checking the data type	`isequal(class(obj.Visible),'char')`
	or
	`ischar(obj.Visible)`
		`isequal(class(obj.Visible),...`
		`'matlab.lang.OnOffSwitchState')`
		or
		`isa(obj.Visible,...`
		`'matlab.lang.OnOffSwitchState')`

Checking the value in a unit test	`verifyEqual(testcase,obj.Visible,'on')`	`verifyEqual(obj.Visible,...` `matlab.lang.OnOffSwitchState.on)`

ActivePositionProperty is not recommended
Still runs

Setting or getting `ActivePositionProperty` on an axes or chart object is not recommended. Use the `PositionConstraint` property instead.

There are no plans to remove `ActivePositionProperty` at this time, but the property is no longer listed when you call the `set`, `get`, or `properties` functions on an axes or chart object.

To update your code, make these changes:

- Replace all instances of `ActivePositionProperty` with `PositionConstraint`.
- Replace all references to the `'position'` option with the `'innerposition'` option.

For example, the following code sets the axes `ActivePositionProperty` to `'position'`.

```
ax = gca;
ax.ActivePositionProperty = 'position';
```

Here is the updated code, which has the same effect.

```
ax = gca;
ax.PositionConstraint = 'innerposition';
```

ChartContainer subclasses assign property values after the setup method runs
Behavior change

When you create an instance of a `ChartContainer` subclass, and pass property name-value pair arguments to the constructor, the property values are assigned after the `setup` method runs. In R2019b, the property values are assigned before the `setup` method runs.

If the `setup` method of your class references the value of a property on the object, you can update your code in either of the following ways:

- Assign a default value for the property when you define it.
- Move the code that references the property to the `update` method.

Calling the ChartContainer.getAxes method returns an axes object as a child of a TiledChartLayout object
Behavior change

When you call the `getAxes` method in a `ChartContainer` subclass, the method now returns an axes object that is a child of a `TiledChartLayout` object. If there are no axes in the chart, `getAxes` creates a Cartesian axes object. The chart no longer has an axes object until you create one by calling the `getAxes` method or one of the axes creation functions: `axes`, `polaraxes`, or `geoaxes`.

As a consequence of these changes, the axes in your chart might not be the current axes. Your code might produce unexpected results if you call the following types of functions within your class methods without specifying the target axes object.

- Plotting functions — For example, `plot`, `scatter`, `bar`, or `surf`
- Functions that modify the axes — For example, `hold`, `grid`, or `title`

In R2019b, the axes object is a child of the chart object, and it is the current axes within the scope of your class methods.

To update your code, specify the axes object as the first input argument when calling plotting functions and functions that modify the axes.

Implementing callbacks on geographic plots disables built-in interactions
Behavior change

Starting in R2020a, when you implement callbacks such as `ButtonDownFcn` on a geographic plot, MATLAB automatically disables built-in interactions.

In previous releases, implementing callbacks did not disable built-in interactions on geographic plots.

Align Distribute Tool will be removed in a future release
Still runs

The Align Distribute Tool will be removed in a future release.

To control the arrangement of multiple plots in a figure, create a tiled chart layout using the `tiledlayout` function instead.

To align or distribute graphics objects within a figure, select **Tools > Align** or **Tools > Distribute** from the figure toolbar instead.

Charting functions return output only when you specify an output argument
Behavior change

The `heatmap`, `geobubble`, `parallelplot`, `scatterhistogram`, `stackedplot`, `wordcloud`, `xline`, and `yline` functions no longer return the chart object as the `ans` variable when you call them without specifying an output argument. This new behavior is consistent with the behavior of most other charting functions.

In previous releases, the functions return the chart object as `ans` by default. If you have code that references a chart object that is stored in the `ans` variable, update your code by assigning the output to a different variable before referencing it.

App Building

uicontextmenu Function: Add and configure context menu components in apps and on the App Designer canvas

You can now create context menus in App Designer apps or in apps created with the `uifigure` function. When you right-click a UI component that has a context menu assigned to it, a list of menu items appears.

In apps created programmatically with the `uifigure` function, create a context menu using the `uicontextmenu` function. Add menu items to it using the `uimenu` function. Then, assign it to a component by setting the `ContextMenu` property of the component to the `ContextMenu` object.

In App Designer, create a context menu and assign it to a component by dragging it from the **Component Library** onto the component. For more details, see “Create and Edit Context Menus”.

uitoolbar Function: Add custom toolbars to apps programmatically

To programmatically add custom toolbars to your App Designer app or your app created with the `uifigure` function, use the `uitoolbar` function. Add push tools or toggle tools to the toolbar using the `uipushtool` or `uitoggletool` functions.

Icon Property: Display SVG, animated GIF, or truecolor image array icons in buttons and tree nodes

The `Icon` property of `Button`, `ToggleButton`, and `TreeNode` objects now supports SVG and animated GIF files and truecolor image array data.

Mouse Pointer: Change the mouse pointer symbol in apps

You can now change the mouse pointer symbol in apps created with the `uifigure` function or in App Designer to options such as `'hand'` or `'crosshair'`, or you can create your own pointer symbol.

For apps created with the `uifigure` function, set the `Pointer` property of the `Figure` object. For apps created with App Designer, select the component in the **Component Browser**. Then, in the **Inspector** tab, select a pointer from the **Pointer** drop-down menu.

To create a custom pointer symbol, programmatically set the `Pointer` property value to `'custom'`, and use the `PointerShapeCData` property to define the symbol. To specify the active pixel of a custom pointer symbol, set the `PointerShapeHotSpot` property.

For more information, see UI Figure Properties.

Graphics Support: Create annotations, brush data, configure data tips, save and copy graphics

You can now create annotations, brush data, configure data tips, and save or copy graphics in App Designer apps or in apps created with the `uifigure` function.

To create annotations, use the `annotation` function.

Use brush mode to mark chart data interactively. Then, you can remove or replace marked data values, or export values to the workspace. To turn on brush mode, select the data brushing button  from the axes toolbar or use the `brush` function to set the brush mode of the figure to `'on'`.

Configure data tips in any of these ways:

- Edit data tip labels by double-clicking them.
- Display multiple pinned (persistent) data tips by clicking more than one data point in the plot.
- Click and drag pinned data tips to move their location with respect to the data point.
- View data tip options by right-clicking on a pinned data tip to display the context menu.

Save graphics displayed in an app using the `exportgraphics` or `copygraphics` function.

GUIDE to App Designer Migration Tool for MATLAB: Migrate GUIDE apps to App Designer in less time and with fewer manual code updates

Improvements to the migration tool significantly reduce the time and number of manual code updates required to get your app running in App Designer. These improvements allow most of your app code to work in App Designer with few or no manual changes required. For more information, see “GUIDE Migration Strategies”.

The GUIDE to App Designer Migration Tool for MATLAB is available through the Add-On Explorer in the MATLAB desktop or through File Exchange on MATLAB Central™ .

App Testing Framework: Perform press gestures with different selection types

The app testing framework supports mouse selection types in `press` gestures that are performed on figures created with the `uifigure` function. To specify the selection type, use the `'SelectionType'` name-value pair argument. For example:

```
fig = uifigure;
testCase = matlab.uitest.TestCase.forInteractiveUse;
testCase.press(fig,'SelectionType','open')
```

Functionality Being Removed or Changed

JavaContainer property will be removed in a future release

Warns

The `JavaContainer` property is undocumented and will be removed in a future release. Update your code to use documented alternatives. For a list of documented functionality that you can use instead, see Recommendations for MATLAB Apps Using Java & ActiveX on `mathworks.com`.

UIContextMenu property of graphics objects and UI components is not recommended

Still runs

Starting in R2020a, using the `UIContextMenu` property to assign a `ContextMenu` object to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The new property can have the same values as the old one.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

Callback property of ContextMenu objects is not recommended
Still runs

Starting in R2020a, using the `Callback` property of a `ContextMenu` object is not recommended. To specify behavior when you interact with a context menu, use the `"ContextMenuOpeningFcn"` property of the `ContextMenu` object instead. The new property can reference callback functions in the same way as the old one.

There are no plans to remove support for the `Callback` property of `ContextMenu` objects at this time. However, this property no longer appears in the list returned by calling the `get` function on a `ContextMenu` object.

Visible and Position properties of ContextMenu objects are not recommended
Still runs

Starting in R2020a, using the `Visible` and `Position` properties to configure a context menu to open at a specific location is not recommended. In apps created with the `uifigure` function, use the `open` function instead.

There are no plans to remove support for the `Visible` and `Position` properties of `ContextMenu` objects at this time. However, these properties no longer appear in the list returned by calling the `get` function on a `ContextMenu` object.

Font size and color of column and row headers in table UI components has changed
Behavior change

Starting in R2020a, table UI components created in App Designer or in figures created with the `uifigure` function display column and row headers in a larger font size and darker font color. For example, this code that creates a table UI component with table array data renders differently in R2020a than it does in R2019b:

```
fig = uifigure;

dates = datetime([2016,01,17; 2017,01,20],'Format','MM/dd/uuuu');
m = [10; 9];
tdata = table(dates,m,'VariableNames',{'Date','Measurement'});

uit = uitable(fig,'Data',tdata);
uit.RowName = 'numbered';
```

R2020a:

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

R2019b:

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

Performance

Live Editor Loop Execution: Improved performance when running loops in live scripts

Loops run significantly faster in live scripts. For example, a live script containing this code, which runs a loop one million times and displays the current loop iteration every ten thousand iterations, is approximately 42x faster.

```
for t = 1:1000000
    if ~mod(t, 10000)
        disp(t)
    end
end
```

The approximate execution times are:

R2019b: 2.291 s

R2020a: 0.054 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

The more iterations a loop contains, the greater the performance improvement becomes.

Live Editor Animation Output: Improved performance when animating plots in live scripts

For-loop animations display faster in live scripts. For example, a live script containing this code, which creates a for-loop animation of a sine wave plot, is approximately 1.3x faster.

```
tic
x = 0:0.1:10*pi;
y = sin(x);
xlim([0 10*pi])
ylim([-1 1])
hold on
p = plot(x(1),y(1));
for k=1:length(x)
    p.XData = x(1:k);
    p.YData = y(1:k);
   drawnow
end
toc
```

The approximate execution times are:

R2019b: 8.875 s

R2020a: 6.633 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU by running the above live script.

datetime, duration, and calendarDuration Data Type Indexing: Improved performance when assigning elements by subscripting

`datetime`, `duration`, and `calendarDuration` subscripted assignment is significantly faster. Performance is now essentially constant with the number of elements in an array.

- For example, when you assign into a `datetime` array with 10^6 elements, performance in R2020a is approximately 25x faster than in R2019b, as shown below.

  ```
  function timingTest()
      dt = datetime + hours(1:1e6);
      indices = randi(1e6,1,10000);
      rhs = NaT;

      tic;
      for i = indices
          dt(i) = rhs;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019b: 0.42 s

  R2020a: 0.017 s

- Similarly, assignment into a `duration` array is faster. For example, when you assign into a `duration` array with 10^6 elements, performance in R2020a is approximately 11x faster than in R2019b.

  ```
  function timingTest()
      d = hours(1:1e6);
      indices = randi(1e6,1,10000);

      tic;
      for i = indices
          d(i) = NaN;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019b: 0.43 s

  R2020a: 0.039s

The code was timed on a Windows 10, Intel® Xeon® W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

These performance improvements occur only when you make subscripted assignments within a function. There is no improvement when subscripting into `datetime`, `duration`, and `calendarDuration` arrays at the command line, or within `try-catch` blocks.

datetime Data Type Format Parsing: Improved performance when parsing format of text inputs

`datetime` parsing performance is significantly faster when parsing the format of text inputs. For example, this code parses the date format of a string using the `datetime` function. The code executes approximately 1.75x faster in R2020a than in R2019b.

```
function timingTest()
   d1 = datetime(2010,1,1:10000);
   s = string(d1,'dd-MMM-uuuu');
   tic
   for i = 1:100, d2 = datetime(s); end
   toc
end
```

The approximate execution times are:

R2019b: 5.44 s

R2020a: 3.10 s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

table Data Type Indexing: Improved performance when assigning elements by subscripting into table variables

`table` subscripted assignment into table variables is significantly faster. Performance is essentially constant with the number of elements in each table variable.

- For example, when you use dot indexing to assign elements to a variable with 10^6 elements, performance in R2020a is approximately 2x faster than in R2019b, as shown below.

  ```
  function timingTest()
      t = table(zeros(1e6,1));
      indices = randi(1e6,1,10000);

      tic;
      for i = indices
          t.Var1(i) = rand;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019b: 1.20 s

  R2020a: 0.59 s

- Similarly, assignment using curly braces is faster. When you assign into three table variables with 10^6 elements, performance in R2020a is approximately 1.2x faster than in R2019b, as shown below.

  ```
  function timingTest()
      t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
  ```

```
        indices = randi(1e6,1,10000);

        tic;
        for i = indices
            t{i,:} = rand;
        end
        toc
    end
```

The approximate execution times are:

R2019b: 8.04 s

R2020a: 6.68 s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

The performance improvement occurs only when you make table subscripted assignments within a function. There is no improvement when subscripting into tables at the command line, or within `try-catch` blocks.

Subscripted Reference: Improved performance for struct arrays stored in a property of an object

The performance of indexing into `struct` arrays that are stored in a property of a MATLAB object has improved. For example, this code executes about 2.3x faster in R2020a:

```
classdef ContainerClass
    properties
        field
    end
end

function out = timingTest

      M = struct('f', 1);
      M(2) = struct('f', 2);
      C = ContainerClass;
      C.field = M;

      tic
      for j = 1:4e5
          out = C.field(1);
      end
      toc

end
```

The approximate execution times are:

R2019b: 1.14 s

R2020a: 0.5 s

The code was timed on a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

imread Function: Improved performance in reading JPEG images

The `imread` function shows improved performance when reading JPEG images. The higher the image resolution (measured in pixels), the greater the performance improvement becomes.

For example, a JPEG image with a resolution of `5120x3840` is read about 2.1x faster in R2020a:

```
function out = testperformance(filename)
for ii = 1:100
    tic
    imread(filename);
    ts(ii) = toc;
end
out = mean(ts)
end
```

The approximate execution times are:

R2019b: 0.46s

R2020a: 0.22s

The code was timed on a Windows 10, Intel Xeon® CPU E5-1650 @ 3.6 GHz test system by calling the function `testperformance`.

readmatrix Function: Improved performance in reading data

The `readmatrix` function shows improved performance when reading matrix data. The larger the matrix to be read, the greater the performance improvement becomes.

For example, a matrix that is 30,000 x 30,000 elements in size is read about 1.1x faster in R2020a:

```
function out = readmatrix_performance()
M = randi(10,30000);
writematrix(M);
for ii = 1:10
tic
readmatrix('M.txt');
ts(ii) = toc;
end
out = mean(ts);
```

The approximate execution times are:

R2019b: 225.21s

R2020a: 198.42s

The code was timed on a Windows 10, Intel Xeon® CPU E5-1650 @ 3.6 GHz test system by calling the function `readmatrix_testperformance`.

ode15s, ode23t, and ode15i Solvers: Improved performance solving differential equations

The `ode15s`, `ode23t`, and `ode15i` solvers show improved performance solving differential equations. The performance improvement gets better as the number of linear systems evaluated by the solver during the solution process increases.

- For `ode15s`, this code executes about 3.5x faster in R2020a:

  ```
  function timingOde15s
  [t,y] = ode15s(@vdp1000,[0 1e5],[2; 0]);
  end
  ```

  The approximate execution times are:

  R2019b: 2.83 s

  R2020a: 0.82 s
- For `ode23t`, this code executes about 4.1x faster in R2020a:

  ```
  function timingOde23t
  [t,y] = ode23t(@vdp1000,[0 1e5],[2; 0]);
  end
  ```

  The approximate execution times are:

  R2019b: 2.92 s

  R2020a: 0.72 s
- For `ode15i`, this code executes about 2.3x faster in R2020a:

  ```
  function timingOde15i
  [y0,yp0] = decic(@weissinger,1,sqrt(3/2),1,0,0);
  for k = 1:100
    [t,y] = ode15i(@weissinger,[1 10],y0,yp0);
  end
  end
  ```

  The approximate execution times are:

  R2019b: 0.51 s

  R2020a: 0.22 s

All timings were performed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingOde15s)
timeit(@timingOde23t)
timeit(@timingOde15i)
```

transpose and ctranspose Functions: Improved performance on large arrays

The performance of the `transpose` and `ctranspose` functions (`.'` and `'` operators) has improved when operating on large arrays. For example, this code executes about 4.4x faster in R2020a when transposing a 10,000-by-10,000 matrix:

```
function timingTest
    rng default
    A = rand(1e4);
    tic
    At = A';
    toc
end
```

The approximate execution times are:

R2019b: 0.61 s

R2020a: 0.14 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.50 GHz test system by calling the function `timingTest`.

ordschur and ordqz Functions: Improved performance operating on large matrices

The `ordschur` and `ordqz` functions show improved performance when operating on square matrices of order 200 or greater. The performance improvement gets better as the matrix gets larger.

- For `ordschur`, this code executes about 11.3x faster in R2020a:

  ```
  function timingOrdschur
  rng default
  A = randn(2000);
  [U,S] = schur(A,'real');
  tic
  [U,S] = ordschur(U,S,'lhp');
  toc
  end
  ```

  The approximate execution times are:

  R2019b: 3.5 s

  R2020a: 0.31 s

- For `ordqz`, this code executes about 8.3x faster in R2020a:

  ```
  function timingOrdqz
  rng default
  A = randn(2000);
  B = randn(2000);
  [A,B,Q,Z] = qz(A,B,'complex');
  tic
  [A,B,Q,Z] = ordqz(A,B,Q,Z,'lhp');
  ```

```
toc
end
```

The approximate execution times are:

R2019b: 18.2 s

R2020a: 2.2 s

All timings were performed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the functions `timingOrdschur` and `timingOrdqz`.

sparse Function: Improved performance constructing sparse matrices

The `sparse` function shows improved performance constructing sparse matrices. The performance improvement applies to all syntaxes of the function and gets better as the constructed matrix gets larger.

For example, this code executes about 4.4x faster in R2020a:

```
function timingSparse
m = 1e5;
n = 1e3;
nz = 1e7;
rng default
i = randi(m,nz,1);
j = randi(n,nz,1);
v = rand(nz,1);

tic
A = sparse(i,j,v,m,n);
toc
end
```

The approximate execution times are:

R2019b: 1.68 s

R2020a: 0.38 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system by calling the function `timingSparse`.

interp1 Function: Faster interpolation for small problem sizes

The `interp1` function shows improved performance for problems with less than about 10,000 sample points. The improvement applies to all interpolation methods.

For example, this code executes about 2.5x faster in R2020a:

```
function timingInterp1
x = 1:100;
v = sin(x/3);
xq = 1:0.5:100;

for k = 1:10000
```

```
    vq = interp1(x,v,xq);
end
end
```

The approximate execution times are:

R2019b: 0.42 s

R2020a: 0.17 s

The code was timed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingInterp1)
```

assert Function: Improved performance for most common use cases

The performance of the `assert` function has significantly improved, making it virtually penalty-free to use assertions in error handling applications. For example, this code executes approximately 300x faster in R2020a:

```
function testAssertPerformance
x = -1;
for i = 1:1e6
    assert(x == -1,'Sample error message.')
end
end
```

The approximate execution times are:

R2019b: 0.30 s

R2020a: 0.001 s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v4 @ 3.60 GHz test system using the `timeit` function:

```
timeit(@testAssertPerformance)
```

There is no performance improvement if `assert` is called with an error message that contains formatting conversion characters, such as those used with the MATLAB `sprintf` function.

nexttile Function: Improved performance when creating several axes in a tiled chart layout

The `nexttile` function shows improved performance when creating several axes in a tiled chart layout. The performance improvement gets better as the number of axes increases.

For example, this code creates 100 axes in a 10-by-10 layout. It executes about 7.2x faster in R2020a.

```
function timingTest
tiledlayout(10,10);
for i = 1:100
    nexttile;
end
end
```

The approximate execution times are:

R2019b: 3.6 s

R2020a: 0.5 s

All timings were performed on a Windows 10, Intel Xeon W-2133 CPU @ 3.60 GHz test system using the `timeit` function:

```
timeit(@timingTest)
```

App Designer Code View: Improved performance when displaying and editing code in App Designer

Displaying and editing code in the App Designer **Code View** editor is faster in R2020a than in R2019b. This affects actions like entering text, creating a new line of code, and adding and deleting functions.

In apps with approximately 2500 lines of code and 4500 lines of code, these types of improvements can be seen:

- Switching from **Design View** to **Code View** is about 1.5x faster in apps with approximately 2500 lines of code, and about 1.6x faster in apps with approximately 4500 lines of code.

  The approximate execution times are:

Release	~2500 Lines of Code	~4500 Lines of Code
R2019b	0.63 s	1.10 s
R2020a	0.42 s	0.70 s

- Entering text in the **Code View** editor is about 2.8x faster in apps with approximately 2500 lines of code, and about 3.9x faster in apps with approximately 4500 lines of code.

  The approximate execution times are:

Release	~2500 Lines of Code	~4500 Lines of Code
R2019b	3.97 s	6.67 s
R2020a	1.44 s	1.72 s

- Creating a new line of code by pressing **Enter** in the **Code View** editor is about 5.8x faster in apps with approximately 2500 lines of code, and about 9.4x faster in apps with approximately 4500 lines of code.

  The approximate execution times are:

Release	~2500 Lines of Code	~4500 Lines of Code
R2019b	1.33 s	3.38 s
R2020a	0.23 s	0.36 s

- Adding and deleting a function in apps with approximately 2500 lines of code are about 1.7x and 2.9x faster, respectively.

  The approximate execution times are:

Release	Add	Delete
R2019b	0.12 s	0.086 s
R2020a	0.07 s	0.030 s

Similarly, in apps with approximately 4500 lines of code, adding and deleting a function are about 1.8x faster and 2.8x faster, respectively.

The approximate execution times are:

Release	Add	Delete
R2019b	0.18 s	0.13 s
R2020a	0.10 s	0.046 s

These actions were timed on a *Windows 10, Intel Xeon® CPU E5-1650 v3 @ 3.5GHz* test system with *NVIDIA® Quadro K620* graphics card.

Graphics Rendering in UI Figures: Improved graphics rendering performance on large data sets in UI figures

In figures created with the `uifigure` function, the graphics rendering performance on large data sets is improved in some cases, such as when displaying these types of plots:

- Surface plots of data larger than a 300-by-300 matrix
- Scatter plots with 20,000 markers or more
- Images where the data stored in the `CData` property is greater than 1 KB

In cases like these, the larger the data set, the greater the performance improvement becomes. These examples show the improvements:

- This code creates the surface plot approximately 5.9x faster in R2020a:

```
function timingTestSurface
num = 10;
tocTimes = zeros(1,num);
fig = uifigure;
ax = uiaxes(fig);
drawnow

for k=1:num
    tic
    s = surface(ax,peaks(500));
    s.EdgeColor = 'none';
    drawnow
    tocTimes(k) = toc;
    delete(s)
    drawnow
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:

R2019b: 3.019 s

R2020a: 0.5102 s

- This code creates the scatter plot approximately 1.1x faster in R2020a:

```
function timingTestScatter
num = 10;
tocTimes = zeros(1,num);
fig = uifigure;
ax = uiaxes(fig);
drawnow
x = linspace(0,3*pi,20000);
y = cos(x) + rand(1,20000);

for k=1:num
    tic
    s = scatter(ax,x,y);
    drawnow
    tocTimes(k) = toc;
    delete(s)
    drawnow
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:

R2019b: 0.0952 s

R2020a: 0.0890 s

- This code displays a 650-by-600-by-3 truecolor image array (1.17 MB) approximately 1.5x faster in R2020a:

```
function timingTestImage
num = 10;
tocTimes = zeros(1,num);
C = imread('ngc6543a.jpg');
fig = uifigure;
ax = uiaxes(fig);
drawnow

for k=1:num
    tic
    im = image(ax,C);
    drawnow
    tocTimes(k) = toc;
    delete(im)
    drawnow
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:

R2019b: 0.0903 s

R2020a: 0.0588 s

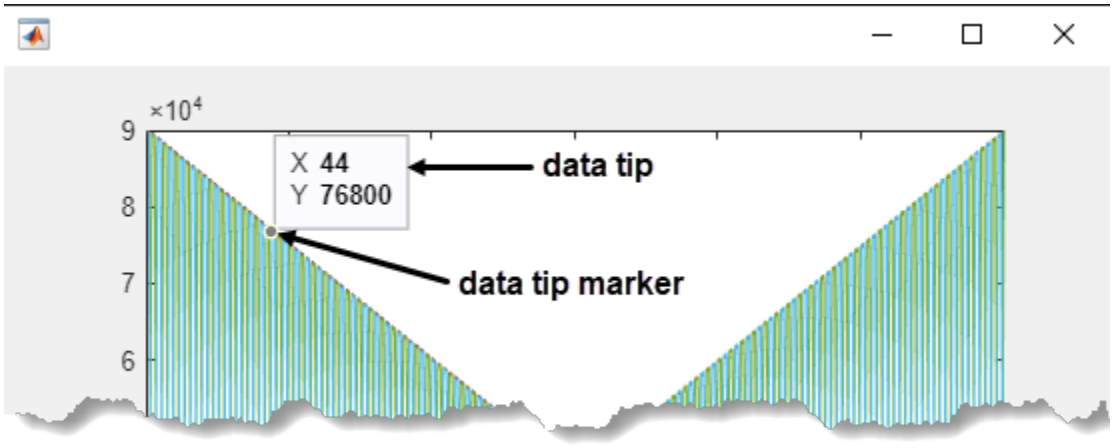
This code was timed on a *Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.6 GHz* test system with *NVIDIA Quadro K620* graphics card by calling the `timingTestSurface`, `timingTestScatter`, and `timingTestImage` functions.

Data Tip Markers: Improved rendering performance of data tip markers in line plots of large data sets created in UI figures and MATLAB Online

In figures created with the `uifigure` function and in MATLAB Online, data tip markers for line plots of large data sets render faster and move more continuously in R2020a than in R2019b. This improvement can be seen when the axes are created with either the `axes` or `uiaxes` function.

For example, on a *Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.6 GHz* test system with *NVIDIA Quadro K620* graphics card, when you move your mouse quickly over the plot lines created by this code, the data tip marker moves more smoothly and tracks your mouse motion more closely in R2020a than in R2019b.

```
fig = uifigure;
ax = axes(fig);
plot(ax,magic(300));
```



Icon Property: Improved rendering performance for buttons and tree nodes with icons

Creating `Button`, `ToggleButton`, and `TreeNode` objects with icons using the `Icon` property is significantly faster. The performance improvement gets better the larger the icon file size, or the more components with icons you have in your app.

For example, this code creates a button with an icon (file size 2.86 MB) approximately 4.6x faster in R2020a:

```
function timingTestButton
r = rand(1000,1000,3);
```

```
imwrite(r,'testimage.png')
num = 10;
tocTimes = zeros(1,num);
fig = uifigure;
drawnow

for k = 1:num
    tic
    btn = uibutton(fig,'Icon','testimage.png');
    drawnow
    tocTimes(k) = toc;
end

disp(['Average Time: ' num2str(mean(tocTimes))])

end
```

The approximate execution times are:

R2019b: 0.705 s

R2020a: 0.154 s

This code was timed on a *Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.6 GHz* test system with *NVIDIA Quadro K620* graphics card by calling the `timingTestButton` function.

Functionality being removed or changed

bench Function: Problem sizes have increased for numerical computation tasks
Behavior change

Starting in R2020a, problem sizes have increased for the numerical computation tasks (LU, FFT, ODE, and Sparse) so that the ranking of machines using `bench` test results are not dominated by the 2-D and 3-D graphics tasks. In previous releases, the 2-D and 3-D tasks take significantly longer to complete compared to the numerical computation tasks and therefore contribute disproportionately to the ranking of machines.

This table shows different task execution times in R2020a using a Windows 10, Intel Xeon W-2133 @ 3.60 GHz test system. The measured values are expressed in seconds.

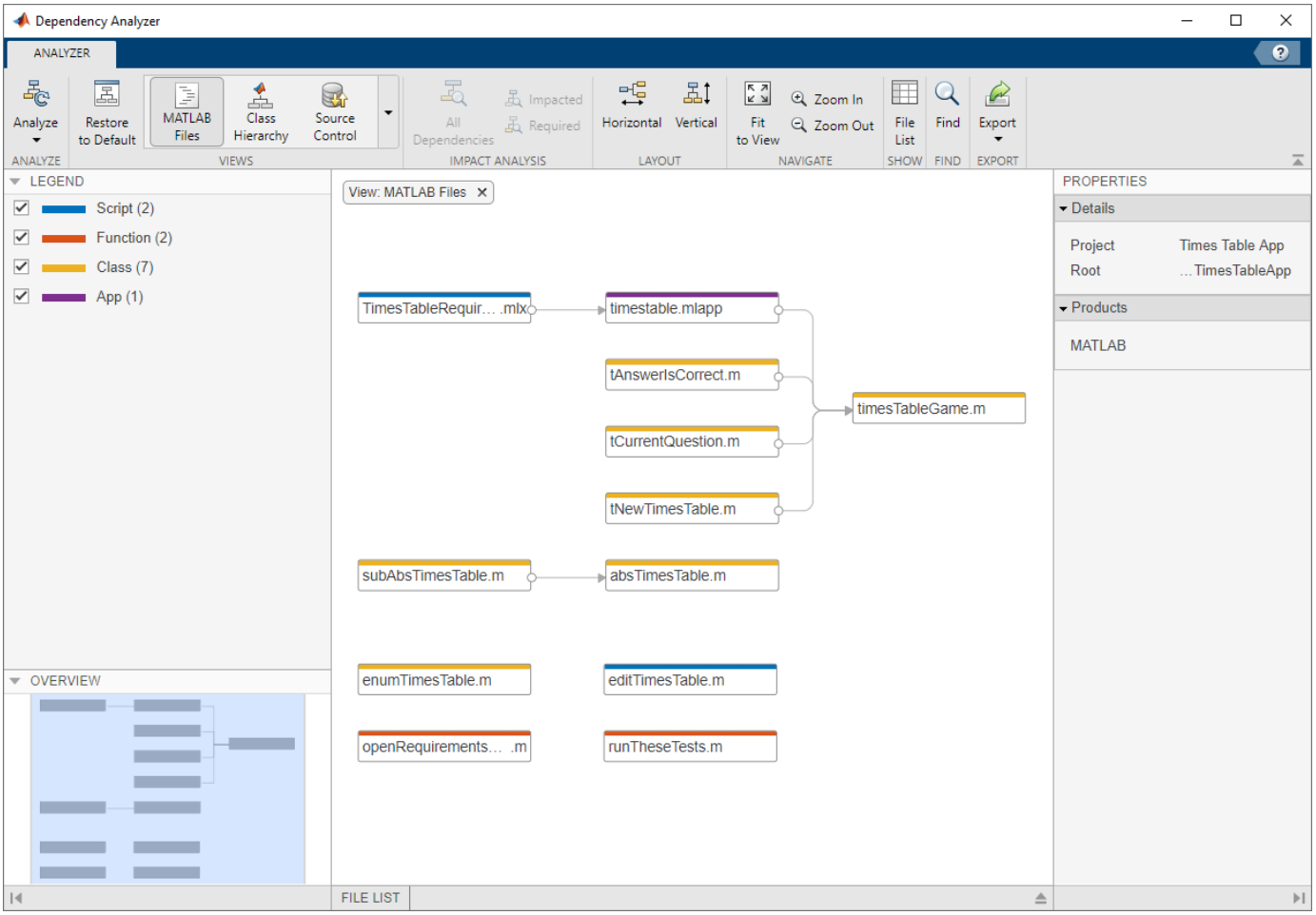
Test	New Problem Sizes	Old Problem Sizes
LU	0.388	0.068
FFT	0.312	0.112
ODE	0.480	0.015
Sparse	0.488	0.103
2-D	0.301	0.307
3-D	0.296	0.303

Software Development Tools

Dependency Analyzer: Improved navigation, filtering, and highlighting for project dependencies

In R2020a, use the new Dependency Analyzer to analyze project dependencies with improved navigation, filtering, and highlighting.

On the **Project** tab, in the **Tools** section, click **Dependency Analyzer**.



Run a dependency analysis to:

- Visualize the project structure and dependencies when you setup or explore a project for the first time. For more details, see “Explore the Dependency Graph, Views, and Filters”.
- Find products and toolboxes required by your design. For more details, see “Find Required Products and Toolboxes”.
- Investigate and resolve problems before sharing, packaging or submitting your project to source control. For more details, see “Investigate and Resolve Problems”.
- Assess the impact of the changes you made on the rest of the design. For more details, see “Find File Dependencies”.

For an example showing how to perform an impact analysis to find and run the tests affected by modified files, see “Perform Impact Analysis with a Project” (Simulink).

Behavior Change

Workflow	R2019b	R2020a
Examine project dependencies and problem files using the file list.	At the top right of the **Dependency Analysis** view, select **Table View**. MATLAB switches to the file list view.	In the Dependency Analyzer, click **File List**. MATLAB opens the file list in the same view as the graph. See “Investigate and Resolve Problems”.
Investigate how two files are related and highlight where the dependency is introduced.	Expand the file in the graph by clicking the arrows next to the file name. To highlight the dependency, double-click the line number in the expanded file.	Select the dependency arrow. In the **Properties** pane, in the **Details** section, you can see the full paths of the files you are examining, the dependency type, and where the dependency is introduced. To highlight where the dependency is introduced, in the **Details** section, click the link under **Impacted**. See “Investigate Dependency Between Two Files”.
Color files in the dependency graph by type, source control status, or label.	On the **Dependency Analysis** tab, in the **Group By** menu, select the option you want.	In the Dependency Analyzer, in the **Views** section, use the different views to explore your project files dependencies. See “Explore the Dependency Graph, Views, and Filters”.

Project Checks: Run all project checks programmatically

You can now use `runChecks` to run all project checks programmatically.

Project API: Get latest Git revision programmatically

You can now programmatically get the latest Git™ revision for every file in your project.

For a project under Git source control, use `currentProject` to create a project object from the currently loaded project.

```
proj = currentProject;
```

Get the latest Git revision of the project file number *`fileNumber`*.

```
proj.Files(fileNumber).Revision
```

For an example on how to get the latest revisions for modified files, see “List Modified Files in Project”.

Unit Testing Framework: Add custom details to TestResult objects

Starting in R2020a, you can add data to the `Details` property of `TestResult` objects when you create your plugins. To append a field to the `Details` structure, use the `append` method of the `matlab.unittest.plugins.plugindata.ResultDetails` class. For more information, see “Write Plugin to Add Data to Test Results”.

Unit Testing Framework: Assert that test session ran with no failure

The `matlab.unittest.TestResult` class has a new method `assertSuccess`, which enables you to assert that no failing conditions were encountered during a test session. For example, run the tests defined in `MyTestClass` and assert that none of them failed.

```
result = assertSuccess(runtests('MyTestClass'));
```

Unit Testing Framework: Run tests from the Live Editor toolstrip

You can now run tests from the MATLAB Live Editor toolstrip. When you open a function-based test file with a `.mlx` extension, the Live Editor toolstrip has options to:

- Run all tests in the file.
- Run the test at your cursor location.

You can customize the test run with options, such as running tests in parallel (which requires Parallel Computing Toolbox) or running tests with a specified level of output detail.

Unit Testing Framework: Generate test reports including test tags

Starting in R2020a, test reports generated using the `TestReportPlugin` class display the test tags for tagged test suite elements. You can generate tagged test reports in DOCX, HTML, and PDF formats. For information about test tags, see “Tag Unit Tests”.

App Testing Framework: Perform press gestures with different selection types

The App testing framework supports mouse selection types in `press` gestures that are performed on UI figures. To specify the selection type, use the `'SelectionType'` name-value pair argument. For example:

```
f = uifigure;
testCase = matlab.uitest.TestCase.forInteractiveUse;
testCase.press(f,'SelectionType','open')
```

Mocking Framework: Add events to mock objects

When creating a mock object, you can add events to the object in addition to properties and methods. To specify the events to mock, use the `createMock` method with the `'AddedEvents'` name-value pair argument. To add events to the mock, the mock object must derive from a handle class. For example:

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock,behavior] = testCase.createMock(?handle,'AddedEvents',{'EventA','EventB'});
```

Mocking Framework: Specify when framework should do nothing

You can specify that the unit testing framework should do nothing each time a mock object method is called or a mock object property is set. Define this behavior with the `matlab.mock.actions.DoNothing` class.

For example, create a mock object with a property `PropA`. Define behavior such that the property is unchanged when it is assigned a value of `0`:

```
testCase = matlab.mock.TestCase.forInteractiveUse;
[mock,behavior] = testCase.createMock('AddedProperties',"PropA");

import matlab.mock.actions.DoNothing
when(behavior.PropA.setToValue(0),DoNothing);
```

Use the mock to assign a nonzero value to `PropA`.

```
mock.PropA = 5

mock =

  Mock with properties:

    PropA: 5
```

Assign a value of `0` to the property. Due to the defined mock object behavior, the value of `PropA` remains unchanged.

```
mock.PropA = 0

mock =

  Mock with properties:

    PropA: 5
```

Functionality being removed or changed

ProfileReport will be removed in a future release

Warns

The `ProfileReport` class will be removed in a future release. Use `CoverageReport` or `CoberturaFormat` instead. Unlike `ProfileReport`, which supports running tests only in serial mode, these two classes can be used to generate reports when tests run in either serial or parallel mode.

To update your code, change instances of `ProfileReport` to `CoverageReport` or `CoberturaFormat`. This table shows an example of how you can update your code.

Before	After

`import matlab.unittest.TestRunner`
`import matlab.unittest.plugins.CodeCoveragePlugin`
`import matlab.unittest.plugins.codecoverage.ProfileReport`
 	
`runner = TestRunner.withNoPlugins;`
`plugin = CodeCoveragePlugin.forFolder('myTests', ...`
`    'Producing',ProfileReport);`
`runner.addPlugin(plugin)`
	`import matlab.unittest.TestRunner`
	`import matlab.unittest.plugins.CodeCoveragePlugin`
	`import matlab.unittest.plugins.codecoverage.CoverageR`
	
	`runner = TestRunner.withNoPlugins;`
	`plugin = CodeCoveragePlugin.forFolder('myTests', ...`
	`    'Producing',CoverageReport);`
	`runner.addPlugin(plugin)`

To create a MATLAB Profiler Coverage Report without specifying a `ProfileReport` format, see “Determine Code Coverage Using the Profiler”.

External Language Interfaces

C++ Interface: MATLAB data type for C++ array and std::vector

MATLAB provides an interface, `clib.array`, which wraps C++ native arrays and `std::vector` types. To create an array of C++ array objects, call the MATLAB `clibArray` function. To convert a MATLAB array to a C++ array, call `clibConvertArray`. For more information, see “MATLAB Object For C++ Arrays”.

For information about creating a MATLAB interface, see “Build MATLAB Interface to C++ Library”.

Compatibility Considerations

By default, beginning in R2020a, MATLAB returns a `clib.array` object instead of the equivalent MATLAB array for primitive types.

To continue to treat a return argument as a MATLAB array, call `clibgen.generateLibraryDefinition` or `clibgen.buildInterface` with the “'ReturnCArrays'” argument set to `false`.

To update your code to use clib arrays, note that when you rebuild the library the MATLAB type for these output arguments changes to `clib.array.`*`libname`*`.`*`classname`*. Also, MATLAB automatically defines more parameters. In general, you still need to provide `SHAPE` information.

C++ Interface: Supported data types

Functionality in a C++ shared library using these types is included in the MATLAB interface to the library.

- `std::shared_ptr`
- Pointer and array data members.
- Double pointers (**) to custom classes used as function or method parameter types. Double pointers to primitive types are not supported.

These types are equivalent to MATLAB `char`:

- `wchar_t`
- `char16_t`
- `char32_t`

These types are equivalent to MATLAB `string`:

- `char *`
- `std::wstring`
- `std::u16string`
- `std::u32string`

For more information, see “MATLAB to C++ Data Type Mapping”. To determine if it is possible to publish an interface to your library, see “Limitations to C/C++ Support”.

C++ Interface: Lifetime management of C++ objects

If a library creates an object, then the library is responsible for releasing the memory. Likewise, if MATLAB creates the object, then MATLAB is responsible for releasing the memory. MATLAB lets you control the lifetime management of objects by specifying `'ReleaseOnCall'` and `'DeleteFcn'` arguments in the library definition file. For more information, see “Lifetime Management of C++ Objects in MATLAB”.

MATLAB Data Array: Support for N-D row-major memory layout

You can create an *N*-D `matlab::data::Array` object with row-major memory layout. Previously, the `createArrayFromBuffer` function created 2-D row-major arrays only. To create a `matlab::data::Array` object with row-major memory layout, set the `createArrayFromBuffer` argument `memoryLayout` to `MemoryLayout::ROW_MAJOR`. To determine the memory layout for an existing `matlab::data::Array` object, call `getMemoryLayout`.

MATLAB COM Server: Register MATLAB without administrative privileges

If you do not have an administrator account on a Microsoft Windows system or you do not start MATLAB with administrative privileges, you can use the `comserver` function to register MATLAB as a COM server for your user account. If you have an administrator account, then you also can call `comserver` to register MATLAB for all users. For more information, see “Register MATLAB as COM Server”. For general information about the MATLAB COM Automation server, see “Calling MATLAB as COM Automation Server”.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2020 for C, C++, and Fortran	Windows
Added	Intel Parallel Studio XE 2020 for Fortran	macOS
Added	Apple Xcode 11.x	macOS
Discontinued	Intel Parallel Studio XE 2017	Windows macOS

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Functionality being removed or changed

`createSoapMessage`, `callSoapService`, and `parseSoapResponse` will be removed
Warns

Consider using `matlab.wsdl.createWSDLClient` instead of the `createSoapMessage`, `callSoapService`, and `parseSoapResponse` functions to communicate with Web services using

Simple Object Access Protocol (SOAP). There is no direct function replacement for the SOAP functions, but when you create a WSDL interface, you have access to the Web service functionality.

createClassFromWsdl will be removed
Warns

The `matlab.wsdl.createWSDLClient` function replaces the `createClassFromWsdl` function to communicate with Web services from MATLAB using Web Services Description Language (WSDL). `matlab.wsdl.createWSDLClient` enables you to specify additional information needed to access the WSDL document. For more information, see `weboptions`.

To get started using `matlab.wsdl.createWSDLClient`, follow these steps.

1. Download supported versions of the Java JDK™ and Apache™ CXF programs. For more information, see “Set Up WSDL Tools”.
2. Set the paths to these programs:

   ```
   matlab.wsdl.setWSDLToolPath('JDK',jdk,'CXF',cxf)
   ```

   where `jdk` is the path to the JDK installation and `cxf` is the path to the CXF program.

To update your code, replace calls to `createClassFromWsdl` with calls to `matlab.wsdl.createWSDLClient`. For example, for a Web service with this URL:

```
url = 'https://examplesite.com/samplewebservice';
```

replace this call to `createClassFromWsdl`:

```
createClassFromWsdl(strcat(url,'?WSDL'))
```

with:

```
matlab.wsdl.createWSDLClient(url)
```

Note `matlab.wsdl.createWSDLClient` does not support RPC-encoded WSDL documents.

Hardware Support

MATLAB Support Package for Ryze Tello Drones: Control Ryze Tello drone from MATLAB and acquire sensor and image data

The MATLAB Support Package for Ryze® Tello Drones is available from release R2020a onwards.

The support package includes functions to pilot Ryze Tello and Ryze Tello EDU drones by sending MATLAB commands to control its direction, speed, and orientation. You can also read flight navigation data such as speed, height, and orientation, capture images, and stream live video into MATLAB from the drone's first-person view (FPV) camera.

Support added for Raspberry Pi 4B model board

You can use the MATLAB Support Package for Raspberry Pi™ Hardware with the Raspberry Pi 4B board.

Deploy deep learning applications on Raspberry Pi hardware

The MATLAB Support Package for Raspberry Pi Hardware now enables you to deploy deep learning applications on the hardware. The deep learning applications continue to run on the Raspberry Pi even if the hardware is disconnected from the computer. To support deployment, the Raspberry Pi functions listed in “Functions Supported for Deployment” (MATLAB Support Package for Raspberry Pi Hardware) are enhanced to generate C++ code. Like any MATLAB function, you can deploy the deep learning application using these steps in “Workflow to Deploy MATLAB Function on Raspberry Pi” (MATLAB Support Package for Raspberry Pi Hardware) with an additional step of creating a deep learning configuration object and assigning it to the `CoderConfig.DeepLearningConfig` property of Raspberry Pi before deployment.

For example, to deploy the deep learning application `raspi_webcam_resnet.m`:

1. Create a configuration object for the Raspberry Pi hardware and set the language of the generated code to C++ using `TargetLang`.

   ```
   t = targetHardware('Raspberry Pi');
   t.CoderConfig.TargetLang = 'C++';
   ```

2. Create an `arm-compute` deep learning configuration object and assign it to the `DeepLearningConfig` property of the Raspberry Pi configuration object.

   ```
   dlcfg = coder.DeepLearningConfig('arm-compute')
   dlcfg.ArmArchitecture = 'armv7';
   t.CoderConfig.DeepLearningConfig = dlcfg;
   ```

3. Deploy the application on the Raspberry Pi hardware.

   ```
   deploy(t,'raspi_webcam_resnet.m')
   ```

Read GPS Data from GPS Receiver Connected to Arduino Hardware

The MATLAB Support Package for Arduino® Hardware enables you to read GPS data from the GPS receiver connected to an Arduino hardware.

Use BNO055 Sensor with Sensor Fusion and Tracking Toolbox, and Navigation Toolbox to Estimate Orientation

You can read acceleration, angular velocity, and magnetic field up to 200Hz in the non-fusion mode of the BNO055 IMU sensor connected to the Arduino hardware. To estimate orientation, you can use the sensor with Sensor Fusion and Tracking Toolbox™, and Navigation Toolbox™.

Enable Code Generation of MATLAB Arduino Functions Inside a MATLAB Function Block for I2C and SPI

In addition to the existing support of ADC, PWM, and Digital Read/Write, you can now generate code for Arduino MATLAB functions inside a MATLAB function block for I2C and SPI.

Functionality being changed or removed

The i2cdev and spidev functions will be removed in R2020a
Warns

Use the `device` instead of `i2cdev` and `spidev` to connect to I2C or SPI devices on Arduino hardware .

The property Pins of servo object will be removed in R2020a
Warns

Use the property `Pin` instead of `Pins` to get the pin number of the Arduino hardware and the Adafruit® Motor Shield V2 for Arduino hardware to which the servo motor is connected. For more information, see Connection to servo motor on Arduino and Connection to servo motor on Adafruit Motor Shield V2 .

The class arduinoio.LibraryBase will be removed in R2020a
Warns

Use the class `matlabshared.addon.LibraryBase` instead of `arduinoio.LibraryBase` for deriving Arduino add-on libraries.

MATLAB support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed will be removed in R2020a
Warns

The support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed in R2020a.

R2019b

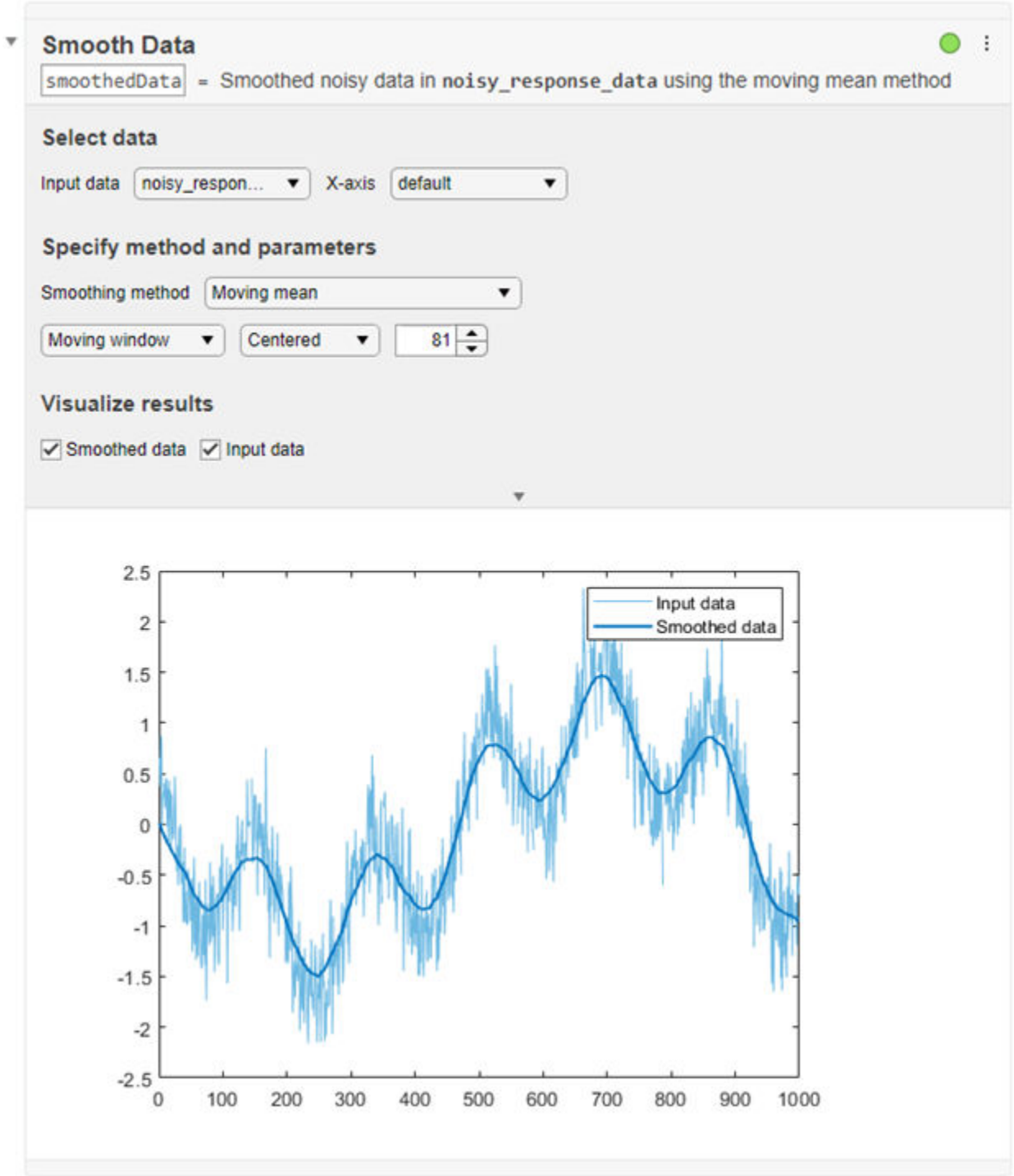
Environment

Live Editor Tasks: Add interactive tasks to live scripts to explore parameters and automatically generate code

Live Editor tasks are app-like interfaces that can be added to a live script to perform a specific set of operations. Use tasks to reduce development time, errors, and time spent plotting. Tasks automatically generate code that becomes part of your live script.

To add a task to a live script, go to the **Live Editor** tab, click **Task**, and select from the available tasks. You also can type the name of the task in a live script code block. As you type, the Live Editor displays possible matches, and you can select and insert the desired task.

For example, add the **Smooth Data** task to a live script to smooth noisy data.

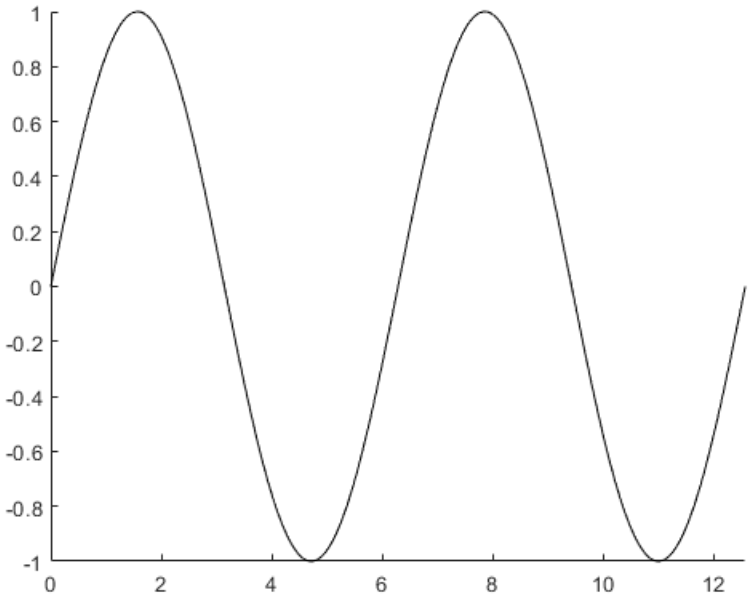


For more information, see Add Interactive Tasks to a Live Script.

Live Editor Output: Animate plots to show changes in data over time

You can include for-loop animations in the Live Editor to show changes in plotted data over time. For example, this code animates a line growing as it accumulates 2,000 data points in the Live Editor. The `drawnow` function displays the changes after each iteration through the loop.

```
h = animatedline;
axis([0 4*pi -1 1])
x = linspace(0,4*pi,2000);

for k = 1:length(x)
    y = sin(x(k));
    addpoints(h,x(k),y);
    drawnow
end
```



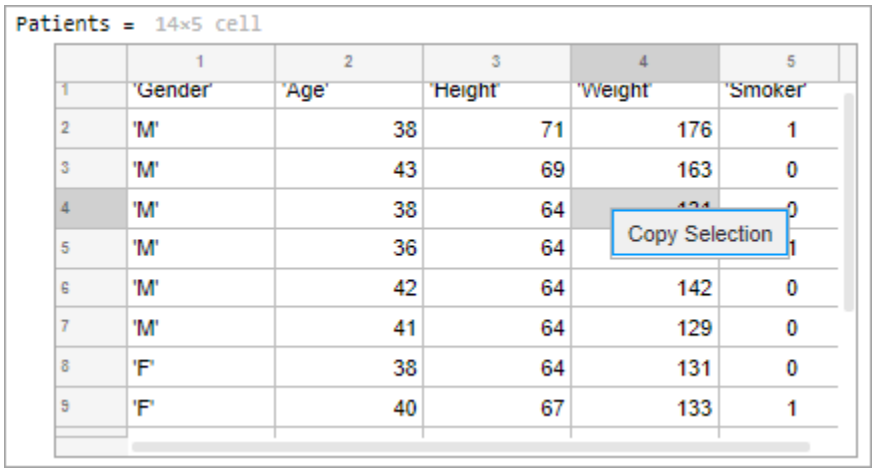
Live Editor Output: Adjust the width of columns in tables

Adjust the width of table columns in the Live Editor by clicking and dragging the column border to the desired width.

Live Editor Output: Scroll through and copy data in arrays such as cell arrays, object arrays, and struct arrays

You can use scroll bars to explore data in cell arrays, object arrays, and struct arrays. You also can scroll through data in string arrays, numeric arrays, categorical arrays, `datetime` arrays, `duration` arrays, and `calendarDuration` arrays.

To copy individual data values in output, select a value, right-click it, and choose the **Copy Selection** option.



Live Editor Export: Customize figure format as well as document paper size, orientation, and margins when exporting

You can programmatically change the resolution and format of images when exporting to PDF and LaTeX in the Live Editor using settings. For example, this code specifies the JPEG figure format and a resolution of 1200 DPI when exporting in the Live Editor for the current session.

```
s = settings;
s.matlab.editor.export.FigureFormat.TemporaryValue = 'jpeg';
s.matlab.editor.export.FigureResolution.TemporaryValue = 1200;
```

You also can programmatically change the paper size, orientation, and margins when exporting to PDF, Microsoft Word documents, and LaTeX in the Live Editor. For example, this code specifies the Legal paper size and landscape page orientation when exporting in the Live Editor for the current MATLAB session.

```
s.matlab.editor.export.pagesetup.PaperSize.TemporaryValue = 'Legal';
s.matlab.editor.export.pagesetup.Orientation.TemporaryValue = 'Landscape';
```

To change the figure format, document paper size, orientation, and margins for an individual export document type, specify the setting for that individual document type. For example, this code specifies a portrait page orientation when exporting to PDF documents in the Live Editor, and a landscape page orientation when exporting to all other document types. Set the personal value instead of the temporary value for the settings to ensure that the values persist across MATLAB sessions.

```
s.matlab.editor.export.pagesetup.pdf.Orientation.PersonalValue = 'Portrait';
s.matlab.editor.export.pagesetup.Orientation.PersonalValue = 'Landscape';
```

For more information, see matlab.editor Settings.

Live Editor Code: Duplicate one or more lines of code

You can reuse one or more lines of code in a live script by duplicating them. To duplicate selected lines, right-click them and select **Duplicate Line(s)**. You also can use the keyboard shortcut **Ctrl+Shift+C** (or **Cmd+Shift+C** on macOS).

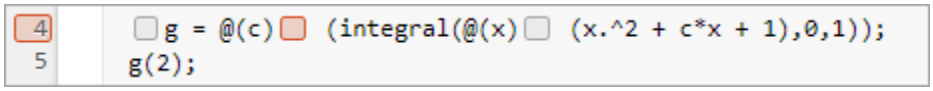
Live Editor Code: Suppress Code Analyzer warning messages

To suppress Code Analyzer warnings for a single line in the Live Editor, right-click a warning and select **Suppress Message... > On This Line**. To suppress the warning for the entire document, select **Suppress Message... > In This File**.

Error messages cannot be suppressed.

Live Editor Debugging: Set breakpoints for anonymous functions

In the Live Editor, you can set breakpoints for anonymous functions. To add a breakpoint, click the gray area to the left of an executable line that contains the anonymous function. MATLAB adds a breakpoint to the line and a disabled breakpoint to the left of each anonymous function in the line. Click the disabled breakpoint to enable it.



Live Editor Internationalization: Add non-English language such as Chinese, Japanese, and Korean characters on Windows and macOS Platforms

You can specify different input methods on Windows and macOS systems to add non-English languages such as Chinese, Japanese, and Korean characters to code and text in the Live Editor.

Add-On Manager: Update MATLAB and other installed add-ons

You can update MATLAB and other installed add-ons to their latest version in the Add-On Manager. To view and install available updates, on the **Home** tab, click **Help > Check for Updates**. The Add-On Manager opens. Select the **Update** tab to view available updates. Click the **Update** button to the right of an available update to install it. For more information, see Get and Manage Add-Ons.

Add-On Manager: Programmatically manage add-ons by name

Enable, disable, and uninstall add-ons programmatically by name using the `matlab.addons.enableAddon`, `matlab.addons.disableAddon`, and `matlab.addons.uninstall` functions.

For example, suppose that you have an add-on called `Random File Name Creator` installed on your system. Disable `Random File Name Creator` by name.

```
matlab.addons.disableAddon("Random File Name Creator")
addons = matlab.addons.installedAddons
```

```
addons =

  1×4 table

            Name                    Version     Enabled                 Identifier
    ____________________________    _______    _________    ____________________________________

    "Random File Name Creator"      "1.0"        false      "75442144-f751-4011-bm0e-32b6fb2f1433"
```

Settings: Create persistent settings for custom apps, toolboxes, and across MATLAB sessions

You can add your own custom settings in MATLAB to store and access data programmatically across a single or multiple MATLAB sessions.

For example, create the settings group `mysettings` and the setting `MyWorkAddress` inside it.

```
s = settings;
addGroup(s,'mysettings','Hidden',false);
addSetting(s.mysettings,'MyWorkAddress','Hidden',false);
s.mysettings.MyWorkAddress.PersonalValue = '3 Apple Hill Drive';
```

You can then use the setting value programmatically in your code.

```
fprintf("I work at %s.\n", s.mysettings.MyWorkAddress.ActiveValue)
```

```
I work at 3 Apple Hill Drive.
```

For more information, see Create Custom Settings. To create settings for custom toolboxes, see Create Factory Settings for Toolboxes.

MATLAB Drive: Share folders and collaborate with others

If you have MATLAB Drive™ Connector installed and actively syncing, you can share folders in your MATLAB Drive directly from MATLAB. To share a folder, right-click the folder in the Current Folder browser and select **Share**.

You can share folders with a view-only link, or invite individual collaborators and set their editing permissions. Invitations can be accepted or declined. After a folder is shared, you can manage the permissions of invited members, rescind invitations, or send additional invitations at any time.

For more information, see Share Folders in MATLAB.

Functionality being removed or changed

Live editor animations enabled by default

Behavior change

Starting in R2019b, for-loop animations in the Live Editor are enabled by default. To disable animations in the Live Editor, set the `matlab.editor.AllowFigureAnimation` setting to `false`:

```
s = settings;
s.matlab.editor.AllowFigureAnimation.PersonalValue = false;
```

Language and Programming

size Function: Find lengths of multiple array dimensions at a time

You can now use a vector dimension argument to query multiple array dimension lengths at a time with the `size` function. For example, `sz = size(A,[1 3])` returns a 1-by-2 row vector containing the lengths of dimensions 1 and 3 of the input array `A`.

matches Function: Determine if input strings are equal

You can determine if two input strings are equal. For more information, see `matches`.

Hexadecimal and Binary Numbers: Specify numbers using hexadecimal and binary literals

You can write numbers using hexadecimal and binary notation. For example, you can write the number 42 as `A = 0x2A` using the `0x` prefix to indicate hexadecimal format. For more information, see Hexadecimal and Binary Values.

Indexing: Use dot indexing into function calls

You can now use dot indexing to index into the result of a function call. MATLAB evaluates the function and then applies the dot indexing operation to the result.

For example, this function creates a structure:

```
function out = createStruct(in)
out = struct("aField", in);
end
```

You can call this function and immediately access the structure field it creates:

```
createStruct(3).aField
```

For more information, see Indexing into Function Call Results.

System object authoring improvements: Property validation support and simplified class inheritance

When you author System objects, you can now use property validation to restrict property values. For information about property validation, see Validate Property Values.

The methods from the `matlab.system.mixin.CustomIcon`, `matlab.system.mixin.Nondirect`, `matlab.system.mixin.Propagates`, and `matlab.system.mixin.SampleTime` mixin classes are now directly included with `matlab.System`. You no longer need to inherit from these mixin classes when authoring System objects.

Function Input Arguments: Declare function input arguments to restrict values

Function argument validation is a way to declare specific restrictions on function input arguments. Using function argument validation, you can constrain the class, size, and other aspects of function input values without writing code in the body of the function to perform these tests. For more information, see Function Argument Validation.

namedargs2cell Function: Convert structure containing name-value pairs to cell array

Convert a scalar structure array that contains name-value data into an interleaved cell array suitable for passing to functions that accept name-value pair cell arrays. This function is typically used with the name-value structure created using function argument validation. For more information, see `namedargs2cell`.

delete, dir, isfile, isfolder, and what Functions: Access web-based storage services like Amazon Web Services and Azure Blob Storage

You can now use the `delete`, `dir`, `isfile`, `isfolder`, and `what` functions to work with remote files and folders. To access remote locations, you must specify the full path using a uniform resource locator (URL). For example, check if a file exists on the specified path in Amazon S3 Cloud.

```
result = isfile('s3://bucketname/path_to_file/my_image.jpg')
```

For more information on setting up MATLAB to access your online storage service, see Work with Remote Data.

Suggested Corrections: Correct errors with two new classes

In addition to the `AppendArgumentsCorrection` class, you can now provide a suggested fix using the `ReplaceIdentifierCorrection` and `ConvertToFunctionNotationCorrection` classes.

- To suggest replacing an identifier in the function call where the exception was thrown, use the `ReplaceIdentifierCorrection` class.
- To suggest the function notation equivalent of the dot notation expression from which the exception was thrown, use the `ConvertToFunctionNotationCorrection` class.

error Function: Provide suggested fix for uncaught exception

You can now use the `error` function to provide a suggested fix when the thrown exception is not caught. To use this functionality, specify the first argument of `error` as an object that suggests the correction. For example, display a suggestion for missing arguments by specifying an `AppendArgumentsCorrection` instance.

```
aac = matlab.lang.correction.AppendArgumentsCorrection(["'InitialCondition'","0"]);
error(acc,msgID,msgText)
```

Functionality being removed or changed

Program files larger than 128 MB or with high complexity produce error
Behavior change

Starting in R2019b, program files larger than approximately 128 MB do not open or run. For files that contain only code (for example, `.m` and `.p` files), this limit affects the file size. For files that store more than just code (for example, `.mlx` files), it affects the size of the code. Running statements larger than 128 MB, either directly in the Command Window or using the `eval` function, also is not supported. In addition, code with high levels of complexity, such as a large number of deeply nested `if` statements, is not supported.

Large program file or statement sizes could result in unpredictable behavior and often occurred when using large portions of code (for example, over 500 lines) to define variables with constant values. To decrease the size of program files, consider defining the variables and saving them in a data file (for example, a MAT-file or `.csv` file). Then you can load the variables instead of executing code to generate them. This adjustment not only decreases the file size of your program but also can increase performance.

Change in rules for function precedence order
Behavior change

Starting in R2019b, MATLAB changes the rules for name resolution, impacting the precedence order of variables, nested functions, local functions, and external functions. These rules are described in Function Precedence Order. For information about the changes and tips for updating your code, see Update Code for R2019b Changes to Function Precedence Order.

- Identifiers cannot be used for two purposes inside a function.
- Identifiers without explicit declarations might not be treated as variables.
- Variables cannot be implicitly shared between parent and nested functions.
- Change in precedence of compound name resolution.
- Anonymous functions can include resolved and unresolved identifiers.

The behavior of the `import` function has changed.

- Change in precedence of wildcard-based imports
- Fully qualified import functions cannot have the same name as nested functions
- Fully qualified imports shadow outer scope definitions of the same name.
- Error handling when import is not found.
- Nested functions inherit import statements from parent functions.

Some repetition arguments for repmat function now produce errors
Behavior change

Starting in R2019b, some `repmat` syntaxes involving nonscalar or empty repetition arguments will produce an error. The following table describes how to update these syntaxes.

Errors	Use Instead
`repmat(A,r1,r2)`, where `r1` and `r2` are row vectors	`repmat(A,[r1 r2])`
`repmat(A,empt)`, where `empt` is an empty array	`repmat(A,1)`
`repmat(A,empt1,empt2)`, where `empt1` and `empt2` are empty arrays	`repmat(A,1)`

Errors	Use Instead
`repmat(A,n,empt)`, where `n` is an integer and `empt` is an empty array	`repmat(A,[n 1])`

Noninteger or complex dimension order arguments for permute function now produce errors
Behavior change

Starting in R2019b, the `permute` syntax `permute(A,dimorder)` produces an error when `dimorder` is a noninteger or complex value. Instead, specify real, positive integer values for `dimorder`.

Compare scalar enumerations in cell arrays to text using the strcmp function
Behavior change

`strcmp` now returns logical `0` or `1` for comparisons of scalar enumerations in cell arrays to text. Previously, `strcmp` always returned `0`.

For example, compare a `matlab.lang.OnOffSwitchState` enumeration in a cell and a character vector.

```
enum = matlab.lang.OnOffSwitchState.off;
TF = strcmp({enum},'off')
```

```
TF =

  logical

   1
```

In previous releases, this comparison using `strcmp` returned `0`.

You can also compare `{enum}` to a cell array of character vectors or to a string array.

```
TF = strcmp({enum},{'on','off'})
```

```
TF =

  1×2 logical array

   0   1
```

However, comparison to strings in cell arrays, such as `strcmp({enum},{"off"})`, always returns `0`. Cell arrays of string arrays are not recommended.

Behavior of nargin for functions that use function argument validation
Behavior change

If the function name input argument to `nargin` refers to a function that uses function argument validation, then the returned value is the number of positional arguments on the function line. This change in behavior can affect code that uses `nargin` with functions that are subsequently updated to use the function argument validation feature. For more information, see nargin in Argument Validation.

Dot-parenthesis syntax will not be allowed in validation function calls
Behavior change in future release

Calling validation functions from `properties` or `arguments` blocks using dot-parenthesis syntax in the function name will be disallowed in a future release. For example, this syntax will be invalid:

```
classdef MyClass
    properties
        A {a.('b').mustBePositive} % Dot-paren not allowed
    end
end
```

Validation function must refer to the property being validated
Behavior change in future release

Property validation functions that do not refer to the property being validated will be disallowed in a future release. For example, in this class definition, the input to the `mustBePositive` function is the constant `10` instead of the property value.

```
classdef MyClass
    properties
        A {mustBePositive(10)} % Property not validated
    end
end
```

To validate the property value, either pass the property name explicitly, or allow MATLAB to pass the property implicitly:

```
classdef MyClass
    properties
        A {mustBePositive}
    end
end
```

Function handles will not be allowed as inputs to property validation functions
Behavior change in future release

Passing function handles to property validation functions will be disallowed in a future release. If you define a validation function that must use a function handle, then define the function handle in the body of the validation function.

For example, it will be an error to pass the function handle `@sin` to the `customValFcn` function:

```
classdef MyClass
    properties
       A {customValFcn(A,@sin)} % Will result in error
    end
end

function customValFcn(A,fh)
    % function body
end
```

Instead, define the function handle in the custom validation function.

```
classdef MyClass
    properties
       A {customValFcn(A)}
    end
end
```

```
function customValFcn(A)
    fh = @sin;
    % function body
end
```

Data Analysis

Live Editor Tasks: Interactively preprocess data and generate code

Use Live Editor tasks to join tables; smooth data; handle outliers and missing data; remove trends; and find local minima, local maxima, and change points. Interactively explore the effects of your changes using generated plots. The tasks also automatically generate code that becomes part of your live script.

MATLAB now offers seven data preprocessing tasks:

- **Join Tables** — Combine two tables using key variables.
- **Smooth Data** — Smooth noisy data.
- **Clean Outlier Data** — Find, fill, or remove outliers.
- **Clean Missing Data** — Find, fill, or remove missing data.
- **Remove Trends** — Remove polynomial trend from data.
- **Find Local Extrema** — Find local maxima and minima.
- **Find Change Points** — Find abrupt changes in data.

To open tasks in the Live Editor, use the **Task** menu on the **Live Editor** tab. For more information, see Add Interactive Tasks to a Live Script.

groupfilter Function: Filter data in a table, timetable, or matrix by group

The `groupfilter` function allows you to filter rows of data by group according to a specified filtering method. For example, given a table `T`, group by variable `Var1` and return only the rows of `T` whose group has more than one element.

```
>> T = table(["A" "B" "C" "A" "A"]',[1 2 3 4 5]')

T =

  5×2 table

    Var1    Var2
    ____    ____

    "A"      1
    "B"      2
    "C"      3
    "A"      4
    "A"      5

>> G = groupfilter(T,'Var1',@(x) numel(x) > 1)

G =

  3×2 table

    Var1    Var2
    ____    ____
```

```
    "A"      1
    "A"      4
    "A"      5
```

datetime Data Type: Detect formats with fractional seconds when converting text that represents dates and times

The `datetime` function allows you to convert text that includes fractional seconds without specifying the `'InputFormat'` name-value pair argument. For example, given text representing a date and time that includes milliseconds, convert the text to a `datetime` value.

```
d = datetime('2019-07-01 12:34:56.123')
```

```
d =

  datetime

   01-Jul-2019 12:34:56
```

While the fractional seconds are detected, the default format for displaying values does not include fractional seconds. To display the fractional seconds, add `'.SSS'` to the `Format` property of `d`.

```
d.Format = 'dd-MMM-uuuu HH:mm:ss.SSS'
```

```
d =

  datetime

   01-Jul-2019 12:34:56.123
```

For more information, see the `Format` property of `datetime`.

table and timetable Data Types: Variable names can have any characters, including spaces and non-ASCII characters

Starting in R2019b, the names of variables in tables and timetables can have any Unicode characters, including spaces and non-ASCII characters. Previously, table and timetable variable names had to be valid MATLAB identifiers. For more information, see `table`, `timetable`, or Access Data in Tables.

This feature has several implications.

- While table and timetable variable names can have any characters, the `table2struct` and `summary` functions do modify variable names that are invalid when you use them as structure field names. The names of fields in a structure must be valid MATLAB identifiers.

  To determine if a name is a valid identifier, use the `isvarname` function.
- You can access any table or timetable variable using dot notation. However, if a variable name is not a valid MATLAB identifier, then you must specify the name as an expression within parentheses following the dot. The expression can be a variable name enclosed in quotation marks, or a function that returns a character vector or string scalar.

  For example, if the table `T` has variables named `'Age'` and `'Blood Pressure'`, then you can use dot notation to access the variables. You can use both of the following syntaxes to access `Age` because it is a valid MATLAB identifier.

```
T.Age
T.('Age')
```

However, to access the variable named `'Blood Pressure'` using dot notation, you must use parentheses because the name is not a valid MATLAB identifier.

```
T.('Blood Pressure')
```

Compatibility Considerations

- Starting in R2019b, table and timetable variable names with leading or trailing whitespace characters are not modified.

  In previous releases, leading and trailing whitespace characters were deleted from variable names when you specified them using the `'VariableNames'` name-value pair argument, or assigned them to the `VariableNames` property.

  This change in behavior affects the `array2table`, `array2timetable`, `cell2table`, `table`, and `timetable` functions, and the `VariableNames` property of tables and timetables. To remove such characters manually, first use the `strtrim` function on the names, then assign them as variable names to the table or timetable.
- Starting in R2019b, MATLAB raises an error if you assign a table variable name that matches a dimension name, or a dimension name that matches a variable name. In previous releases, MATLAB raised a warning and modified the names so they were unique.

tall Arrays: Operate on tall arrays with more functions, including setdiff, xcorr, and outerjoin

The functions listed in this table now support tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

`isregular`	`vartype`
`outerjoin`	`xcorr`
`setdiff`	`xcov`
`setxor`	

In addition, some functions have removed limitations with tall arrays.

Functions	Added Support
`min` and `max`	`min` and `max` now fully support tall arrays. *Previously, you could not specify multiple outputs.*
`conv`, `conv2`, and `convn`	`conv`, `conv2`, and `convn` now support more mixtures of tall arrays and in-memory arrays in the inputs. *Previously, the second input `B` could not be a tall array.*

Functions	Added Support
`innerjoin`	`innerjoin` now fully supports tall arrays. *Previously, you could only specify one output argument, and one of the inputs was required to be an in-memory array.*
`intersect`	`intersect` now allows both inputs to be tall arrays. *Previously, one of the inputs was required to be an in-memory array.*

tall Arrays: Avoid running out of memory due to temporary copies of data

If you convert an in-memory array `A` into a tall array using `T = tall(A)`, then MATLAB avoids creating copies of the underlying data when you perform subsequent operations on `T`, which relaxes the memory requirements of many operations. This approach is useful if, for example, you have 8 GB of RAM and perform an operation on a 5 GB array that normally requires a temporary copy of the data.

Previously, the syntax `tall(A)` for an in-memory array `A` did not avoid creating temporary copies of data, and was bound by the same memory requirements as in-memory arrays.

Data Import and Export

detectImportOptions Function: Specify the type of import options for delimited or fixed-width text files

When detecting import options for a text file, the `detectImportOptions` function now enables you to specify the type of the text file as `'delimitedtext'` or `'fixedwidth'`. For more information, see `detectImportOptions`.

table and timetable Data Types: Read and write tabular data that has variable names containing any characters, including spaces and non-ASCII characters

Starting in R2019b, you can read and write tables and timetables containing variable names that have any Unicode characters, including spaces and non-ASCII characters. Previously, table and timetable variable names had to be valid MATLAB identifiers.

To read tabular data that contains arbitrary variable names, such as variable names with spaces and non-ASCII characters, set the `PreserveVariableNames` parameter to `true`.

```
T = readtable('myFile.xls','PreserveVariableNames',true)
```

The following functions support reading and writing of tabular data containing variable names that contain any characters:

- **Import Tool**, `readtable`, `readtimetable`, and `parquetread`
- `detectImportOptions`, `spreadsheetImportOptions`, `delimitedTextImportOptions`, and `fixedWidthImportOptions`
- `tabularTextDatastore`, `spreadsheetDatastore`, and `parquetDatastore`
- `writetable`, `writetimetable`, and `parquetwrite`
- `save` and `load`

If you save a table or timetable to a MAT-file (using the `save` function), then the MAT-file contains both original and modified names for any variables whose names are not valid identifiers. If you load the MAT-file into R2019b or later (using the `load` function), then the table or timetable uses original variable names. If you load it into R2019a or earlier, then the table or timetable uses the modified variable names.

For more information, see `table`, `timetable`, or Access Data in Tables.

sheetnames Function: Get names of worksheets from spreadsheet file

Use the `sheetnames` function to get the names of all worksheets from a spreadsheet file. For more information, see `sheetnames`.

VideoReader Object: Read frames in videos using frame index or time

The `VideoReader` object supports interchangeable access to video frames using frame index or time. Therefore, you can now use `read` and `readFrame` interchangeably.

Previously, you could use only one type of access at a time. Attempting to read frames interchangeably using `read` and `readFrame` resulted in an error. For example, the following code would result in an error.

```
vidObj = VideoReader('xylophone.mp4');
frame20 = read(vidObj,20); % read using frame index
frame21 = readFrame(vidObj) % read next frame using CurrentTime
```

```
Cannot call 'READFRAME' method after querying the
NUMBEROFFRAMES property or using the READ method.
Recreate the object to use 'READFRAME' method.
```

However, starting in R2019b, you can use all the methods and properties of the `VideoReader` object interchangeably between accessing frames using frame indices and accessing frames using time. For example, create a `VideoReader` object and read the 20th frame using the `read` method. Next, use the `readFrame` method to get the 21st frame.

```
vidObj = VideoReader('xylophone.mp4');
frame20 = read(vidObj,20);
frame21 = readFrame(vidObj);
whos frame20 frame21
```

```
  Name           Size                Bytes  Class    Attributes

  frame20      240x320x3            230400  uint8
  frame21      240x320x3            230400  uint8
```

For more information, see `VideoReader`.

VideoReader Object: Improved performance in generated code with row-major layout

For large video files, the generated code for the `VideoReader` object with a row-major layout option shows improved performance. For example, the `timingTest` function shows about a 4x speed-up on a `H.264` video file with a resolution of `1280x720`.

```
function [t, data] = timingTest(fileName)
  vidObj = VideoReader(fileName);
  data = cell(20,1);
  tic;
  for cnt = 1:20
    data{cnt} = readFrame(vidObj);
  end
  t = toc;
end
```

Generate code for the `timingTest` function with the row-major flag. The `codegen` command creates a function `timingTest_mex` with the C and C++ generated code.

```
codegen timingTest -args {coder.typeof('', [1 inf])} -rowmajor
```

For a `H.264` video file with a resolution of `1280x720`, the execution times are:

R2019a: 4.04s

R2019b: 0.95s

The code was timed on a Windows 10, Intel Xeon® CPU W-2133 @ 3.6 GHz test system by calling the function `timingTest_mex`. The higher the video file resolution (measured by frame size), the greater the performance improvement becomes.

Import Tool: Generate simpler code when importing from fixed-width text files

Import Tool now operates consistently across different platforms and generates code that is easy to read when importing fixed-width text files. For example, the generated code contains `readtable` and `FixedWidthImportOptions`, which makes the code simpler to read and use. For more information, see Read Text File Data Using Import Tool.

save Function: Save workspace variables to a MAT-file version 7 without compression

Previously, the `save` command, when saving workspace variables to a version 7 MAT-file, used compression as the default (and the only) option. Now, `save` supports the `'-nocompression'` option for MAT-file version 7.

By default, saving a variable `myVariable` compresses the data and saves it to a version 7 MAT-file. The `-v7` flag is optional.

```
save myFile.mat myVariable –v7
```

To save `myVariable` without compression, use:

```
save myFile.mat myVariable –v7 –nocompression
```

For more information, see `save`.

xmlread Function: Prevent reading of XML files that contain DOCTYPE declarations

You can prevent reading XML files that contain DOCTYPE declarations by setting the `'AllowDoctype'` name-value pair to `false`. For more information, see the `xmlread` reference page.

imread Function: Supports reading specified images from PGM, PBM, or PPM file formats

The `imread` function supports reading specified images from multi-image PGM, PBM, or PPM file formats. For more information, see the `imread` reference page.

Scientific File Format Libraries: CFITSIO Library upgraded to version 3.450

The CFITSIO library is upgraded to version `3.450`.

Scientific File Format Libraries: LibTIFF Library upgraded to version 4.0.10

The LibTIFF library is upgraded to version `4.0.10`.

RESTful Functions: Support for authentication

The RESTful web services functions `webread`, `websave`, and `webwrite` also support Digest authentication. For more information, see the `weboptions 'Username'` argument.

For the list of supported authentications for RESTful functions, see “Server Authentication”.

Compatibility Considerations

The RESTful functions `webread`, `webwrite`, and `websave` now adhere more closely to the Internet Engineering Task Force (IETF®) document RFC 7617 for Basic authentication. As a result, MATLAB might error when a RESTful function communicates with a server that proactively expects Basic authentication but does not return a 401 challenge response.

To update your code, see How do I preemptively include a Basic Authentication header when working with "webread"/"webwrite"/"websave" in MATLAB R2019b?

tcpclient, read, and write Functions: Generate C and C++ code

The `tcpclient`, `read`, and `write` functions support C and C++ code generation using MATLAB Coder.

Bluetooth Low Energy Interface: Support for scanning and interacting with peripheral devices

You can use MATLAB commands to perform the following operations:

- Scan for nearby peripheral devices and view advertising data using the `blelist` function
- Connect to peripheral devices using the `ble` function
- Access device characteristics and descriptors using the `characteristic` and `descriptor` functions
- Read device characteristic data and descriptor data using the `read` function
- Write to device characteristics and descriptors using the `write` function
- Enable and disable notification or indication for a characteristic using the `subscribe` and `unsubscribe` functions

For more information, see Bluetooth Low Energy Communication.

Serial Port Devices: New functions and properties

The serial port interface has a new set of functions and properties. The existing functionality still runs, but new function names and properties are recommended. The new interface has improved performance.

Get started with the new interface by viewing a list of all serial ports on your computer using `serialportlist`.

```
list = serialportlist
```

```
list =

  1×4 string array

    "COM1"    "COM3"    "COM4"    "COM8"
```

Then, create a `serialport` object, write data to the device, and read from it.

```
s = serialport("COM8",115200);
write(s,1:5,"uint32")
read(s,5,"uint32");
```

For more information, see Serial Port Devices.

Compatibility Considerations

For more information about updating your code to use the recommended functionality, see Transition Your Code to `serialport` Interface.

Functionality being removed or changed

Import Tool does not support importing text data as cell array of character vectors
Behavior change

Previously, **Import Tool** provided an option to import text data as a cell array of character vectors. Starting in R2019b, **Import Tool** does not support this. Instead, the `Import Tool` app imports text data as string arrays.

To preserve the previous behavior, convert the imported text data to a cell array of character vectors using the `cellstr` function.

UseExcel parameter for spreadsheets
Behavior change

The default setting for `UseExcel` on Windows systems with Excel® installed is `false`.

The `UseExcel` parameter appears on these spreadsheet functions: `readtable`, `readtimetable`, `readmatrix`, `readcell`, `readvars`, `writetable`, `writetimetable`, `writematrix`, and `writecell`.

To preserve the previous behavior, update calls to these functions to specify the `UseExcel` parameter as `true`. For example, for the `readtable` function, update your code as follows.

```
T = readtable(filename,'UseExcel',true)
```

xlsfinfo function is not recommended
Still runs

The `xlsfinfo` function is not recommended. Use the function `sheetnames` instead. There are no plans to remove `xlsfinfo` at this time. The `sheetnames` function has these advantages over the `xlsfinfo` function:

- Support for sheet names containing non-ASCII characters
- Better cross-platform support and performance
- Ability to work with remotely stored data

This table shows a typical use of `xlsfinfo` and how to update your code to use `sheetnames`.

Not Recommended	Recommended
`[~,sheets] = xlsfinfo('myData.xls')`	`sheets = sheetnames('myData.xls')`

NumberOfFrames property of the VideoReader object is not recommended
Still runs

The `NumberOfFrames` property of the `VideoReader` object is not recommended. Use the property `NumFrames` instead. Update all instances of `NumberOfFrames` with `NumFrames`. There are no plans to remove the `NumberOfFrames` parameter at this time.

Array of VideoReader objects is not supported

Creating an array of `VideoReader` objects is not supported. Update your code to remove arrays of `VideoReader` objects. For example, the following code returns an error.

```
v = VideoReader('xylophone.mp4');
v(end+1) = VideoReader('xylophone.mpg');
```

```
Array formation and parentheses-style indexing
with objects of class 'VideoReader' is not allowed.
Use objects of class 'VideoReader' only as scalars
or use a cell array.
```

Tiff object for writing certain TIFF files

Writing TIFF images with certain combinations of photometric configuration and the number of samples per pixel is not recommended. The value of `SamplesPerPixel` must be equal to the sum of `Photometric` color channels and the `ExtraSamples` specified in the `Tiff` object. For more information, see `Tiff` and `write`.

imwrite function does not support writing of indexed PNG files that have insufficient colormap entries

Starting in R2019b, for writing indexed PNG files, you must specify a colormap with enough entries to interpret the image correctly. For more information, see the `imwrite` reference page.

imfinfo function returns information on multiple images from PGM, PBM, and PPM files
Behavior change

Previously, for the PGM, PBM, and PPM file formats, the `imfinfo` function returned a single 1-by-1 structure. The structure contained information about only the first image, even if the file contained multiple images in it.

Starting in R2019b, if the PGM, PBM, and PPM files have multiple images, then `imfinfo` returns a structure array containing information on multiple images in the file. For instance, for a PGM file containing `M` images, `imfinfo` returns a `1`-by-`M` structure array containing information corresponding to all the images in the file.

This table shows how to update your code to get the `Height` (or any other property) of the first image of a multi-image PBM, PGM, or PPM file.

Not Recommended	Recommended
`info = imfinfo('MultiImgFile.pgm')` `info.Height`	`info = imfinfo('MultiImgFile.pgm')` `info(1).Height`

For more information, see `imfinfo`.

web Function
Behavior change

The `web` function now opens external sites using your system browser by default. Previously, the `web` function opened external sites using the MATLAB browser. Using the system browser is recommended when opening external sites.

To use the MATLAB browser as the default browser for external sites, go to the **Home** tab, and in the **Environment** section, click **Preferences**. Select **MATLAB > Web** and in the **System Web browser** section, clear the **Use system web browser when opening links to external sites (recommended)** option.

Web services use system certificates
Behavior change

The default value for the `CertificateFilename` option in the `weboptions` function and the `matlab.net.http.HTTPOptions.CertificateFilename` property is `'default'`. If the value is `'default'`, then MATLAB uses system certificates.

Previously by default, MATLAB used the PEM certificate file that ships with MATLAB.

seriallist function is not recommended
Still runs

`seriallist` is not recommended. Use `serialportlist` instead. See Transition Your Code to `serialport` Interface for more information about using the recommended functionality.

serial function is not recommended
Still runs

`serial` and its object properties are not recommended. Use `serialport` and its properties instead.

This example shows how to connect to a serial port device using the recommended functionality.

Functionality	Use Instead

`s = serial("COM1");`
`s.BaudRate = 115200;`
`fopen(s)`
	`s = serialport("COM1",115200);`

See Transition Your Code to `serialport` Interface for more information about using the recommended functionality.

Mathematics

makima Function: Perform modified Akima cubic Hermite interpolation

The `makima` function performs modified Akima cubic Hermite interpolation, similar to the `'makima'` interpolation method of `griddedInterpolant`, `interp1`, `interp2`, `interp3`, and `interpn`. The modified Akima cubic Hermite interpolation method has these properties:

- It is C^1 continuous.
- It produces fewer undulations than `spline`, but the result is not as aggressively flattened as `pchip`.
- Unlike `pchip`, it supports N-D arrays.
- Unlike `spline`, it does not produce overshoots.

Graphics

Chart Container Class: Develop your own class of charts

Define your own class of charts by creating a subclass of the `matlab.graphics.chartcontainer.ChartContainer` base class. If you write scripts or functions for creating specialized visualizations and share them with others, consider creating a class implementation. Creating a class enables you to:

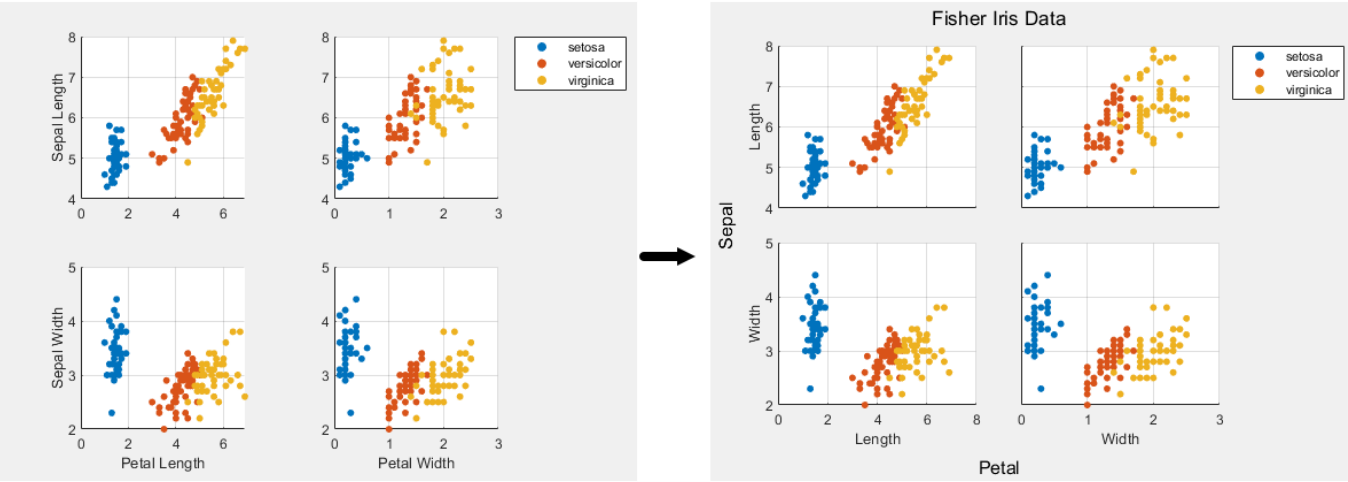
- Provide a convenient interface for your users — When users want to customize an aspect of your chart, they can set a property rather than having to modify and rerun your graphics code. Users can modify properties at the command line or inspect them in the Property Inspector.
- Encapsulate algorithms and primitive graphics objects — You implement methods that perform calculations and manage the underlying graphics objects. Organizing your code in this way allows you to hide implementation details from users.

The `ChartContainer` base class supports charts that have a single Cartesian axes. When you define a chart that derives from this base class, instances of your chart are members of the graphics object hierarchy. As a result, your charts are compatible with many aspects of the graphics system. For example, the `gca` and `findobj` functions can return instances of your chart. For more information, see Chart Development Overview.

tiledlayout and nexttile Functions: Create configurable layouts of plots in a figure

Use the `tiledlayout` and `nexttile` functions to lay out a tiling of multiple plots within a figure. The configuration options include:

- Control over the spacing between the plots and around the edges of the layout
- An option for a shared title at the top of the layout
- Options for shared *x*- and *y*-axis labels
- An option for displaying a shared axes toolbar
- An option to control whether the tiling has a fixed size or variable size that can reflow



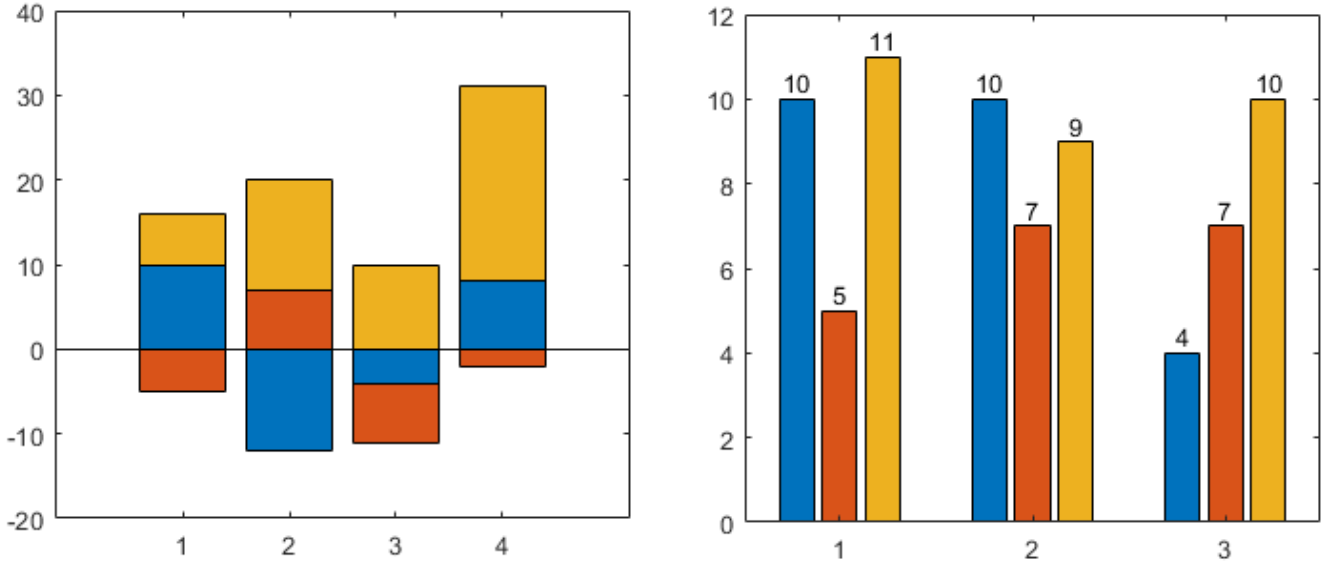
colororder Function: Control the colors in plots

Control the color scheme of your plots by calling the `colororder` function. When you change the color scheme using this function, the change has an immediate effect on your existing plots. If you call the `colororder` function for an empty figure, the new color scheme persists in the figure when you call a plotting function.

Bar Charts: Create bar charts with improvements for stacking and locating the tips of bars

The `bar` and `barh` functions have these improvements:

- Stacked groups of bars display negative bars below zero, rather than overlapping the bars.
- You can locate the tips of bars by getting the values of the `XEndPoints` and `YEndPoints` properties on the `Bar` object. The locations are useful for adding labels to the tips of bars.
- Both functions now accept more combinations of scalar, vector, and matrix inputs.



Data Tips: Create and customize data tips

Create data tips on objects with a `DataTipTemplate` property using the `datatip` function. You can create a data tip by specifying the exact coordinates of the data point, the approximate coordinates of the data point, or the index of the data point within the plotted data set. Create a data tip between plotted data points by setting the `SnapToDataVertex` property to `'off'`.

In the Live Editor, generate code for interactively pinned data tips by clicking the **Update Code** button. This generated code recreates the data tip the next time you run the live script.

Customize the content of data tips on additional charts, including `Contour`, `Patch`, `Quiver`, `Bar`, and `Image` objects with a `DataTipTemplate` property. For more information, see Create Custom Data Tips.

dataTipInteraction Function: Pin data tips at cursor location

Create data tip interactions that pin data tips to the cursor location, rather than the nearest data point, by setting the `SnapToDataVertex` property of the data tip interaction object to `'off'`:

For more about customizing axes interactions, see Control Chart Interactivity.

Axes Toolbar: Save or copy contents of axes as image

To save or copy the contents of a set of axes or a tiled chart layout, hover over the **Export** icon and select an option from the drop-down menu. The available options depend on the content of the axes.

- Save to an image or PDF file by selecting **Save As**.
- Copy to the clipboard as a PNG image by selecting **Copy as Image**.
- Copy to the clipboard as a vector graphic for PDFs by selecting **Copy as Vector**.

When you create a custom toolbar using the `axtoolbar` function, include a drop-down menu with save and copy options by specifying the `buttons` argument as `'export'`.

parallelplot Function: Zoom, pan, and rearrange coordinates interactively

The parallel coordinates plot has new options for interacting with data:

- Zoom — Use the scroll wheel to zoom.
- Pan — Click and drag the plot to pan.
- Rearrange coordinates — Click and drag a coordinate tick label to move the corresponding coordinate ruler to a different position.

Property Inspector: Update axis tick values and labels using clipboard data

You can update the tick values and labels that appear along an axis by copying and pasting data (for example, from a spreadsheet) into the property inspector. To paste copied data, open the property inspector and navigate to the ticks property that you want to edit. Then, open the drop-down menu by clicking and select **Paste**.

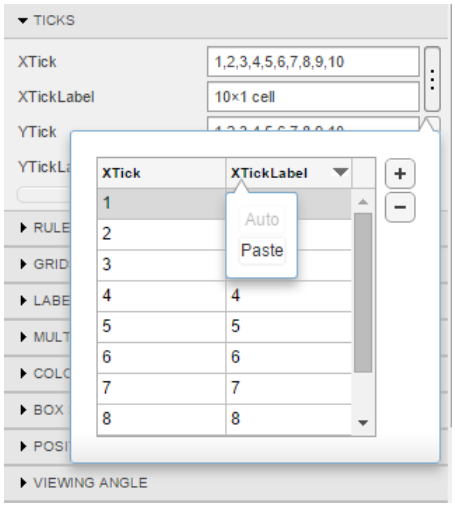


Image Interpolation: Select an interpolation method for displaying images

Display images using either nearest neighbor or bilinear interpolation. MATLAB uses interpolation when it displays a scaled or rotated version of an image.

When you call the `image`, `imagesc`, or `imshow` functions, specify the `Interpolation` property name-value pair with either `'nearest'` or `'bilinear'` as the value. All images use `'nearest'` by default.

For example, this code displays an image using the `imagesc` function with bilinear interpolation:

```
I = imread('peppers.png');
imagesc(I,'Interpolation','bilinear')
```

legend Function: Create unlimited legend entries and specify categorical arrays

The `legend` function has these improvements:

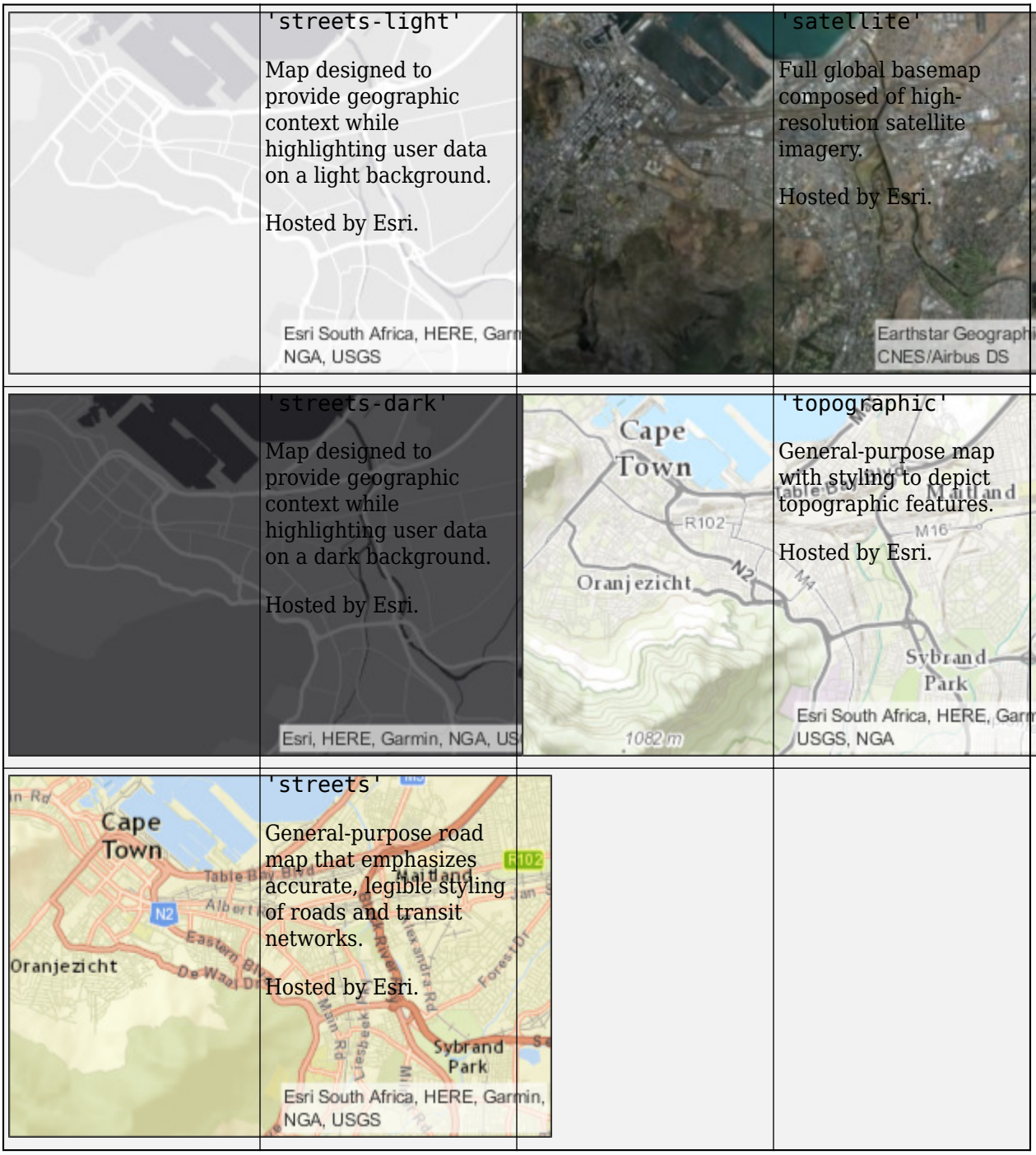
- The legend can display an unlimited number of entries when you specify the `subset` argument. In previous releases, legends display up to 50 entries only.
- You can specify label values as categorical arrays in addition to cell arrays and string arrays.

pcolor Function: Specify categorical, datetime, and duration data

The `pcolor` function now accepts categorical, `datetime`, and `duration` arrays for `X` and `Y`.

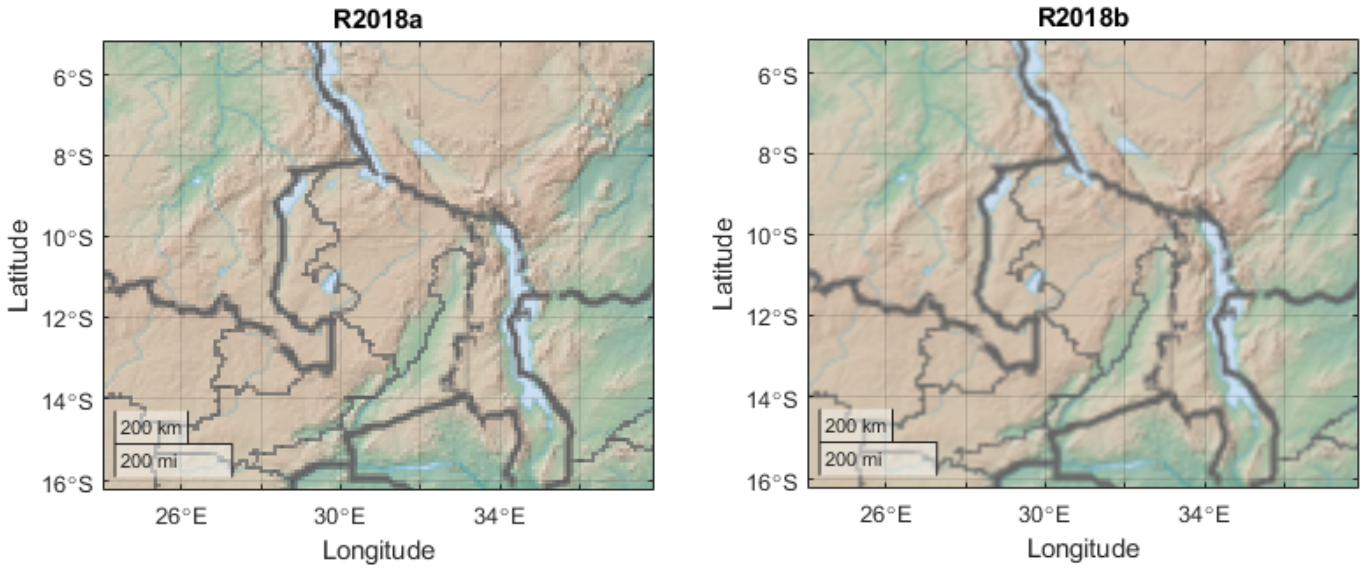
Geographic Plots: Plot data on high-zoom-level basemaps

Plot data on high-zoom-level basemaps hosted by Esri®. For more information about Esri, see https://www.esri.com. Specify a basemap using the `geobasemap` function or by setting the `Basemap` property of the `GeographicAxes` or `GeographicBubbleChart` object.



Geographic Plots: Create plots with improved basemap appearance

Starting in R2018b, basemaps within geographic plots have an improved visual appearance at noninteger zoom levels. For example, the `ZoomLevel` property of these geographic bubble charts is 4.995.



Geographic Axes: Display animations using comet or animatedline

Display animations on geographic axes using `comet` or `animatedline`. For more information, see Line Animations.

Geographic Bubble Charts: Create charts with improved layout

Geographic bubble charts have an improved visual appearance, including use of figure space and placement of ticks and tick labels.

When you resize a geographic bubble chart, font sizes and spacing between elements in the chart automatically adjust to provide the best possible presentation for the new size. Changing the `FontSize` property of a geographic bubble chart disables the automatic resizing of the fonts.

Functionality being removed or changed

Changing ColorOrder or LineStyleOrder on the axes affects existing plots immediately

Behavior change

If you change the axes `ColorOrder` or `LineStyleOrder` properties after plotting into the axes, the colors and line styles in your plot change immediately. In previous releases, the new colors and line styles affect only subsequent plots, not the existing plots.

For example, this code plots two lines before changing the value of the `ColorOrder` property. In R2019a and previous releases, the colors of the two lines do not change when you change the `ColorOrder` property.

```
plot([0 1],[1 0])
hold on
plot([0 1],[2 0])
ax = gca;
ax.ColorOrder = [1 0 0; 0 1 0];
```

Starting in R2019b, the lines change immediately to use the new `ColorOrder` or `LineStyleOrder` property values. If you want to prevent the change, set either the axes `LineStyleOrderIndex` or `ColorOrderIndex` property to any value (such as its current value) before changing the `ColorOrder` or `LineStyleOrder` property.

```
plot([0 1],[1 0])
hold on
plot([0 1],[2 0])
ax = gca;

% Preserve R2019a behavior
ax.ColorOrderIndex = ax.ColorOrderIndex;

ax.ColorOrder = [1 0 0; 0 1 0];
```

Indexing scheme for ColorOrder and LineStyleOrder might change plot colors and line styles
Behavior change

In R2019a and previous releases, plots that visualize multiple sets of data rely on an indexing scheme to select the colors and line styles. The indexing scheme generally works well, but it does not allow you to change the colors and line styles in a plot after you create it.

Starting in R2019b, there is a new indexing scheme that enables you to change the colors and line styles of existing plots. MATLAB always applies this indexing scheme to objects such as `Line`, `Scatter`, and `Bar`. As a result, your code might produce plots that cycle through the colors and line styles differently than in previous releases.

One example of the new behavior is when you create a line with a specific color, and then create another line without specifying a color. In the following code, the first call to the `plot` function includes a third argument that specifies the color as red. The second call to the `plot` function does not specify a color. In R2019a, the first line is red, and the second line is blue.

```
% Plot two lines
hold on
plot([0 1],[1 0],'red')
plot([0 1],[2 0])
```

If you run the preceding code in R2019b, the first line is red and the second line is orange. To preserve the original color, set either the axes `LineStyleOrderIndex` or `ColorOrderIndex` property to any value (such as its current value) before plotting into the axes.

```
% Preserve R2019a behavior
ax = gca;
ax.ColorOrderIndex = ax.ColorOrderIndex;

% Plot two lines
hold on
plot([0 1],[1 0],'red')
plot([0 1],[2 0])
```

Predefined colormaps have 256 colors by default
Behavior change

The predefined colormaps, such as `parula`, `jet`, and `winter`, now have 256 colors by default.

If you have code that depends on a predefined colormap having 64 colors, specify the number of colors when you set the colormap for the figure, axes, or chart. For example, `colormap(parula(64))` sets the figure's colormap to the 64-color `parula` colormap.

Alternatively, you can change the default colormap for all figures within your MATLAB session:

```
set(groot,'defaultFigureColormap',parula(64))
```

Pie charts display zero values
Behavior change

When you call the `pie` or `pie3` function and specify data that contains zero values, your pie chart shows the zero values and corresponding labels. If you call either function with an output argument, the output includes objects for each zero value.

In previous releases, the functions omit the zero values from the chart and do not return any objects that correspond to those values. If you do not want to display zero values or return the corresponding objects, then remove the zeros from your data.

Using the axis function to set axes limits no longer changes the view of the plot box
Behavior change

When you call the `axis` function to set limits, the plot box no longer changes its view in these situations:

- When the plot box is in a 2-D view, and you pass a six-element vector to the `axis` function. To preserve the behavior of previous releases, call `view(3)` after calling the `axis` function.
- When the plot box is in a 3-D view, and you pass a four-element vector to the `axis` function. To preserve the behavior of previous releases, call `view(2)` after calling the `axis` function.

Default basemap for geographic plots is now 'streets-light'
Behavior change

Starting in R2019b, the default basemap for `GeographicAxes` and `GeographicBubbleChart` objects is `'streets-light'`. The `'streets-light'` basemap requires Internet access.

In previous releases, the default basemap was `'darkwater'`. This basemap is included with MATLAB and does not require Internet access.

If you do not have reliable access to the Internet, you can use the `'darkwater'` basemap or download a selection of basemaps onto your local system using the Add-On Explorer. For more information, see Access Basemaps in MATLAB.

Turning on interaction modes disables mouse interactions for geographic bubble charts
Behavior change

Starting in R2019b, turning on an interaction mode in a figure disables mouse interactions for geographic bubble charts in the figure. For example, if you turn on zoom mode using the `zoom` function, then you can no longer use the mouse to zoom or pan within the geographic bubble chart. Other interaction modes include pan, rotate, data cursor, or brush mode.

To zoom or pan within a geographic bubble chart, turn off interaction modes within the figure and zoom or pan using the built-in mouse interactions.

Geographic bubble charts display tick labels using seconds

Behavior change

Starting in R2019b, the tick label format for geographic bubble charts is degrees, minutes, and decimal seconds (DMS) rather than degrees and decimal minutes (DM).

App Building

uistyle Function: Create styles for rows, columns, or cells in a table UI component

You can create different styles for specific rows, columns, or cells in a table UI component using the `uistyle` and `addStyle` functions. For example, you can make the cells in a specific column red with italic font. To retrieve styles that have been applied, get the `StyleConfigurations` property of the `Table` object. To remove a style from a table UI component, use the `removeStyle` function.

Cell styles in table UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

uigridlayout Function: Configure grid rows and columns to adjust automatically to fit components

Grid layouts can automatically adjust to the minimum size needed to fit the components that are in it. To configure grid rows and columns to fit components dynamically, specify `'fit'` as the value of the `RowHeight` or `ColumnWidth` properties for specific rows and columns in the `GridLayout` object. For example, setting `'RowHeight'` to `{'fit',50,'1x'}` specifies that the height of the first row is tightly fit around the components, the second row is fixed at 50 pixels, and the third row uses the remaining vertical space.

This feature is especially useful when you are creating rows and columns of text-based components because when you use `'fit'`, you don't have to know how tall or wide to make the rows and columns of the grid layout manager. The `'fit'` option for row height and column width is also useful if your app is translated to another language or runs on different platforms.

The `uigridlayout` function is supported only in App Designer apps and in figures created with the `uifigure` function.

uitable Function: Sort table UI components interactively when using logical, numeric, string, or cell arrays

You can interactively sort table UI components when the `Data` property contains logical data, numeric data, string data, or cell array data. To sort table UI components that contain these data types, set the `ColumnSortable` property of the `Table` object. To update visualizations based on how a table UI component was sorted, also use the `DisplayData` property and a `DisplayDataChangedFcn` callback function.

Sortable columns in table UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

uihtml Function: Embed HTML, JavaScript, or CSS content in apps and on the App Designer canvas

To embed HTML, JavaScript®, or CSS in your app, call the `uihtml` function or, in App Designer, drag an HTML UI component from the **Component Library** onto the canvas.

HTML UI components are supported only in App Designer apps and in figures created with the `uifigure` function.

App Designer: Convert components in a UI figure or container from pixel-based positioning to a grid layout manager

You can convert the children of a UI figure or container from pixel-based positioning to being positioned by a grid layout manager. To use a grid layout manager where you were previously using pixel-based positioning, drag a grid layout from the **Component Library** onto the canvas or into an existing container component, like a panel. Alternatively, right-click the canvas or container component and select **Apply Grid Layout**.

The grid layout manager automatically creates rows and columns to accommodate the components, and preserves their approximate positions. When you add a grid layout, the component hierarchy updates in the **Component Browser**.

App Designer: Convert an existing app into an auto-reflowing app

To convert an existing app into an auto-reflowing app, expand the **Convert** drop-down menu in the **Canvas** tab, and select **2-Panel App with Auto-Reflow** or **3-Panel App with Auto-Reflow**.

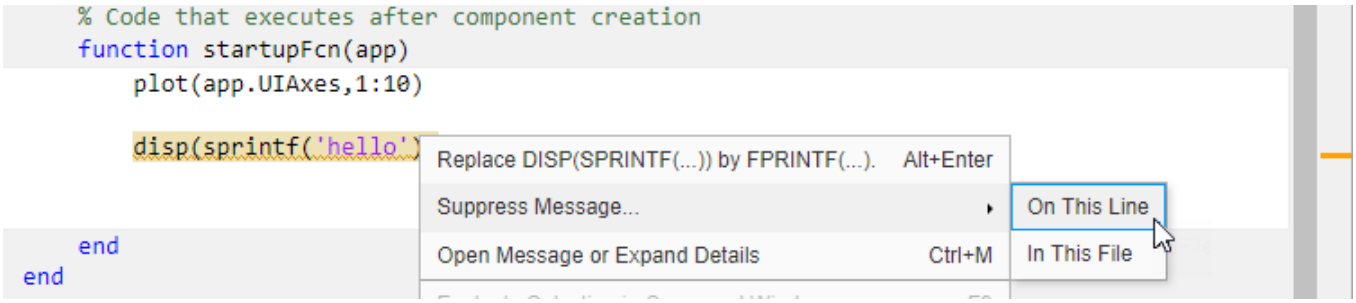
Auto-reflowing apps automatically resize and reflow content based on screen size, screen orientation, and platform. Use apps with auto-reflow if you expect to run or share your apps across multiple environments or desktop resolutions. For more details, see Apps with Auto-Reflow.

App Designer: Suppress Code Analyzer warning messages

In the App Designer **Code View** editor, you can suppress Code Analyzer warnings for a single line or for the entire app file. To suppress warnings, right-click a warning and, from the context menu, select:

- **Suppress Message... > On This Line**
- **Suppress Message... > In This File**

For example,



Error messages, however, cannot be suppressed.

App Designer: Open App Designer from the MATLAB toolstrip

To open App Designer from the MATLAB toolstrip, click the **Design App**  button in the **Apps** tab.

App Testing Framework: Perform gestures on polar axes and UI images

The App testing framework supports gestures on more UI components.

- Perform hover and press gestures in tests on polar axes.
- Perform press gestures in tests on UI images.

For example, perform interactive gestures on `PolarAxes` object `pax` and `Image` object `im`.

```
fig = uifigure;
pax = polaraxes(fig,'ThetaAxisUnits','radians');
im = uiimage(fig,'ImageSource','membrane.png','Position',[10 10 100 100]);
testCase = matlab.uitest.TestCase.forInteractiveUse;
testCase.hover(pax);
testCase.press(pax,[pi/2 0.5]);
testCase.press(im);
```

For more information, see the `hover` and `press` reference pages.

Functionality being removed or changed

GUIDE will be removed in a future release

Still runs

The GUIDE environment and the `guide` function will be removed in a future release.

After GUIDE is removed, existing GUIDE apps will continue to run in MATLAB but will not be editable using the drag-and-drop environment in GUIDE. To continue editing an existing GUIDE app and help maintain its compatibility with future MATLAB releases, use one of the suggested migration strategies listed in the table.

App Development	Migration Strategy	How to Migrate
Frequent or ongoing development	Migrate your app to App Designer	Use the GUIDE to App Designer Migration Tool for MATLAB on `mathworks.com`
Minimal or occasional editing	Export your app to a single MATLAB file to manage your app layout and code using MATLAB functions	Open the app in GUIDE and select **File > Export to MATLAB-file**

App Designer is the recommended app development environment in MATLAB. To create new apps, use App Designer and the `appdesigner` function instead.

To learn more about migrating apps, see GUIDE Migration Strategies.

For more information about App Designer, go to Comparing GUIDE and App Designer on `mathworks.com`.

javacomponent function and JavaFrame property will be removed in a future release
Warns

The `javacomponent` function and the `JavaFrame` figure property are undocumented and will be removed in a future release. Update your code to use documented alternatives. For a list of documented functionality that you can use instead, see Recommendations for Java and ActiveX Users on `mathworks.com`.

actxcontrol, actxcontrollist, and actxcontrolselect functions will be removed in a future release
Warns

The `actxcontrol`, `actxcontrollist`, and `actxcontrolselect` functions will be removed in a future release. Update your code to use alternate functionality. For a list of functionality that you can use instead, see Recommendations for Java and ActiveX Users on `mathworks.com`.

Support for running deployed web apps in Internet Explorer has been removed
Errors

Support for running deployed web apps in Internet Explorer® has been removed. Use the current versions of Google Chrome™ (recommended), Safari, Firefox®, or Microsoft Edge® to run deployed web apps instead.

For more information on supported web app browsers, see Supported Browsers and Platform Incompatibilities (MATLAB Compiler).

Text alignment and font size have changed in table column and row headers
Behavior change

Starting in R2019b, table UI components created in App Designer or in figures created with the `uifigure` function have a different visual appearance when they contain certain kinds of data. Column and row headers of table UI components that contain numeric, logical, string, or cell array data have these visual differences compared to previous releases:

- Smaller font size
- Column headers are left-aligned instead of center-aligned
- Row headers are center-aligned instead of left-aligned

For example, this code that creates a table UI component with mixed cell array data renders differently in R2019b than it does in R2019a.

```
fig = uifigure;

d = {'Male',52,true;'Male',40,true;'Female',25,false};

uit = uitable(fig,'Data',d);
uit.ColumnName = {'Gender','Age','Authorized'};
```

R2019b:

	Gender	Age	Authorized
1	Male	52	☑
2	Male	40	☑
3	Female	25	☐

R2019a:

	Gender	Age	Authorized
1	Male	52	☑
2	Male	40	☑
3	Female	25	☐

Starting in R2019a, the same visual differences apply to column and row headers in table UI components that contain table array data. For example, this code that uses a table array to display `datetime` values in a table UI component renders differently in R2019a than it does in R2018b.

```
fig = uifigure;

dates = datetime([2016,01,17; 2017,01,20],'Format','MM/dd/uuuu');
m = [10; 9];
tdata = table(dates,m,'VariableNames',{'Date','Measurement'});

uit = uitable(fig,'Data',tdata);
uit.RowName = 'numbered';
```

R2019a:

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

R2018b:

	Date	Measurement
1	01/17/2016	10
2	01/20/2017	9

Performance

table Data Type Indexing: Improved performance when assigning elements by subscripting into large table variables

`table` subscripted assignment into large table variables is significantly faster. Performance is now essentially constant with the number of elements in each table variable.

- For example, when you use dot indexing to assign elements to a variable with 10^6 elements, performance in R2019b is approximately 40x times faster, as shown below.

  ```
  function timingTest()
      t = table(zeros(1e6,1));
      indices = randi(1e6,1,10000);

      tic;
      for i = indices
          t.Var1(i) = rand;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019a: 47.83 s

  R2019b: 1.20 s

- Similarly, assignment using curly braces is faster. For example, when you assign into three table variables with 10^6 elements, performance in R2019b is approximately 18x faster.

  ```
  function timingTest()
      t = table(zeros(1e6,1), ones(1e6,1), nan(1e6,1));
      indices = randi(1e6,1,10000);

      tic;
      for i = indices
          t{i,:} = rand;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019a: 156.39 s

  R2019b: 8.51 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling each version of the `timingTest` function.

The larger the table variables are, the greater the performance improvement becomes. However, the performance improvement occurs only when you make table subscripted assignments within a function. There is no improvement when subscripting into tables at the command line, or within try-catch blocks.

datetime, duration, and calendarDuration Data Type Indexing: Improved performance when assigning elements by subscripting into large arrays

`datetime`, `duration`, and `calendarDuration` subscripted assignment into large arrays is significantly faster. Performance is now essentially constant with the number of elements in an array.

- For example, when you assign into a `datetime` array with 10^6 elements, performance in R2019b is approximately 106x times faster, as shown below.

  ```
  function timingTest()
      dt = datetime + hours(1:1e6);
      indices = randi(1e6,1,10000);
      rhs = NaT;

      tic;
      for i = indices
          dt(i) = rhs;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019a: 49.00 s

  R2019b: 0.46 s

- Similarly, assignment into a `duration` array is faster. For example, when you assign into a `duration` array with 10^6 elements, performance in R2019b is approximately 106x times faster.

  ```
  function timingTest()
      d = hours(1:1e6);
      indices = randi(1e6,1,10000);

      tic;
      for i = indices
          d(i) = NaN;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019a: 48.66 s

  R2019b: 0.46 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling each version of the `timingTest` function.

The larger the arrays are, the greater the performance improvement becomes. However, the performance improvement occurs only when you make subscripted assignments within a function. There is no improvement when subscripting into `datetime`, `duration`, and `calendarDuration` arrays at the command line, or within try-catch blocks.

datetime Data Type Indexing: Improved performance when referring or assigning to date and time components of datetime arrays

Subscripted references and assignments to components of `datetime` arrays is significantly faster.

- For example, when you refer to a component of a `datetime` array with 10^4 elements, performance in R2019b is approximately 25x times faster, as shown below.

  ```
  function timingTest()
      dt = datetime + hours(1:1e4);
      indices = randi(1e4,1,10000);

      tic;
      for i = indices
          x = dt.Hour(i);
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019a: 12.97 s

  R2019b: 0.52 s

- Similarly, assignment into a component of a `datetime` array is faster. For example, when you assign into a component of a `datetime` array with 10^4 elements, performance in R2019b is approximately 32x times faster.

  ```
  function timingTest()
      dt = datetime + days(1:1e4);
      indices = randi(1e4,1,10000);

      tic;
      for i = indices
          dt.Hour(i) = 0;
      end
      toc
  end
  ```

  The approximate execution times are:

  R2019a: 22.51 s

  R2019b: 0.70 s

The code was timed on a Windows 10 system with a 3.6 GHz Intel Xeon W-2133 CPU by calling each version of the `timingTest` function.

The larger the arrays are, the greater the performance improvement becomes. However, the performance improvement occurs only when you make subscripted assignments within a function. There is no improvement when subscripting into `datetime` arrays at the command line, or within try-catch blocks.

uitable Function: Faster performance when data type is numeric, logical, or a cell array of character vectors

Tables created with the `uitable` function have better rendering performance, and higher frame rates while scrolling when they contain certain kinds of data. The improvements occur when the `Data` property contains numeric data, logical data, or a cell array of character vectors. The table must be parented to a figure created with the `uifigure` function, or one of its child containers.

Tables containing these data types render up to 40% faster, and interaction performance (like scrolling) is up to 75% faster. For example, on a test system, this code that uses numeric data renders the table faster in R2019b than in previous releases.

```
rows = 10000;
columns = 25;
ndata = randi(30,[rows columns]);
fig = uifigure;
uit = uitable(fig,'Data',ndata);
```

unzip and gunzip Functions: Improved performance when extracting contents of zip files and GNU zip files

Extracting the contents of zip files and GNU zip files using `unzip` and `gunzip` is significantly faster when extracting files on network drives.

- For example, when you extract the contents of the example zip file `myarchive.zip` with a file size of 53 MB on a network drive, performance in R2019b is approximately 1.5x faster, as shown below.

  ```
  function timingTest()
      unzip myarchive.zip;
  end
  ```

  The approximate execution times are:

  R2019a: 3.06 s

  R2019b: 2.03 s
- Similarly, when you extract the contents of the example GNU zip file `myotherarchive.gz` with a file size of 27 MB on a network drive, performance in R2019b is approximately 2x faster, as shown below.

  ```
  function timingTest()
      gunzip myotherarchive.gz;
  end
  ```

  The approximate execution times are:

  R2019a: 37.22 s

  R2019b: 18.22 s

The code was timed on a Windows 10 test system with a 3.6 GHz Intel Xeon CPU E5-1650 CPU across a Gigabit Ethernet connection using the `timeit` function:

```
timeit(@timingTest)
```

Results vary depending on several factors, including connection speed and whether the network files are cached on the system.

Software Development Tools

Unit Testing Framework: Run tests in parallel with your custom plugins

You can now run unit tests in parallel when you extend the `TestRunner` instance with your custom plugins. For more information, see Run Tests in Parallel with Custom Plugin.

Unit Testing Framework: Validate count in string constraints

The `ContainsSubstring`, `IsSubstringOf`, and `Matches` constraints can now count a string scalar or character vector for a given number of times. To specify the number of occurrences, use the `'WithCount'` parameter. For example:

```
import matlab.unittest.constraints.*
testCase = matlab.unittest.TestCase.forInteractiveUse;
testCase.verifyThat('This is long',ContainsSubstring('is','WithCount',2))

Verification passed.

testCase.verifyThat('Gain Main Rain',Matches('[GMR]ain','WithCount',2))

Verification failed.
    ----------------------
    Framework Diagnostic:
    ----------------------
    Matches failed.
    --> Actual count does not match expected count.

        Actual Count:
             3
        Expected Count:
             2

    Actual char:
        Gain Main Rain
    Regular Expression:
        [GMR]ain
```

Performance Testing Framework: Visually compare two TimeResult arrays

The `matlab.perftest.TimeResult` class has a new method `comparisonPlot`, which enables you to visually compare the time measurement results of two equal-sized sets of performance tests.

App Testing Framework: Perform gestures on polar axes and images

The app testing framework supports gestures on more UI components.

- Perform `hover` and `press` gestures in tests on polar axes.
- Perform press gestures in tests on images.

Projects: Delete project definition files

You can now use matlab.project.deleteProject to easily stop managing your folder with a project and delete all related project definition files without affecting the remaining files.

Compare Git Branches: Show differences and save copies

In a project under Git source control, you can now select any two revisions and examine file differences. You can show differences between two development branches and save a copy of the selected file on either branch. See Compare Branches.

Functionality being removed or changed

Character vectors are no longer equivalent to enumerations in qualifications

Behavior change

Starting in R2019b, actual and expected values in qualifications must have the same type when the expected value is an enumeration of a handle class. For example, consider this enumeration class:

```
classdef MyClass < handle
    enumeration
        X
        Y
    end
end
```

The following test fails because `'X'` does not represent the enumeration `MyClass.X`:

```
testCase = matlab.unittest.TestCase.forInteractiveUse;
testCase.verifySameHandle('X',MyClass.X)
```

```
Verification failed.
    ----------------------
    Framework Diagnostic:
    ----------------------
    verifySameHandle failed.
    --> Values do not refer to the same handle.
    --> Value must be a handle object. It is of class "char".
    --> Classes do not match.
            Actual Value class           : [char]
            Expected Handle Object class : [MyClass]

    Actual char:
        X
    Expected Handle Object:
      MyClass enumeration

        X
```

In previous releases, the test passed because MATLAB treated `'X'` as a representation of the expected enumeration. This change of behavior affects tests using the `IsSameHandleAs` constraint class or the following qualification methods: `verifySameHandle`, `assumeSameHandle`, `assertSameHandle`, `fatalAssertSameHandle`, `verifyNotSameHandle`, `assumeNotSameHandle`, `assertNotSameHandle`, and `fatalAssertNotSameHandle`.

External Language Interfaces

C++ Interface: Options for publishing C++ interface library

MATLAB automatically renames classes, functions, enums, and member functions with C++ names that are invalid in MATLAB using the `matlab.lang.makeValidName` function. For example, MATLAB converts the class name `_myclass` in library `mylib` to `x_myclass`. As of R2019b, you can modify `x_myclass` in the library definition file. For example, you can change the name to `myclass`. When you use the class in MATLAB, type `clib.mylib.myclass`. Renaming C++ namespaces or the MATLAB package is not supported.

To specify the shape for object pointer types as scalar for all functions in a library, use the name-value pair argument `'TreatObjectPointerAsScalar'` when building the library. To specify the shape for `const char *` pointer types as scalar for all functions, use the `'TreatConstCharPointerAsCString'` argument.

To provide a list of macro definitions, use the name-value pair argument `DefinedMacros` when building the library. To provide a list of macro cancellations, use the `UndefinedMacros` argument.

For more information, see `clibgen.generateLibraryDefinition` and `clibgen.buildInterface`.

C++ Interface: nullptr supported as output argument

As of R2019b, the “C++ interface returns type-specific empty values for nullptr” on page 2-50. To test for fundamental `nullptr` types, call the `isempty` function. To test for `nullptr` objects, call the `clibIsNull` function.

C++ Interface: Read-only (const) object support

As of R2019b, the “C++ interface treats read-only objects like C++” on page 2-50. To determine if a C++ object is read-only, call the `clibIsReadOnly` function.

Java Interface: JRE version 1.8.0_202 support

The MATLAB interface to Java supports JRE™ version 1.8.0_202, providing improved security and access to new Java features.

Out-of-Process Execution of C++ MEX Functions: Customize environment variables

To customize the environment of a MEX host process that you use to execute a MEX function, call `mexhost` with the `"EnvironmentVariables"` argument.

HTTP Web Services: Server authentication support for NTLM and Kerberos protocols

The HTTP interface also supports these protocols for server authentication.

- Windows — NTLM and Kerberos
- Linux and macOS — NTLM

For more information, see Server Authentication.

HTTP Web Services: Timeout options

MATLAB has new timeout options for transmitting messages using the HTTP interface.

- `DataTimeout` — timeout in seconds between packets on the network
- `KeepAliveTimeout` — how long the connection to the server stays open after an initial connect, enabling multiple successive messages to be sent over the same connection
- `ResponseTimeout` — seconds to wait for the header of the response from the server after sending the last packet of a request

For more information, see `matlab.net.http.HTTPOptions`.

Python Interface: Execute Python functions out of process

Run Python® functions in processes that are separate from the MATLAB process. For more information, see Out-of-Process Execution of Python Functionality. Use this mode to call functions in third-party libraries that are not compatible with MATLAB.

Python Interface and Engine: Version 3.5 support discontinued

Support for Python version 3.5 is discontinued.

Compatibility Considerations

To ensure continued support for your applications, upgrade to a supported version of Python, version 3.6 or 3.7.

Perl 5.30.1: MATLAB support on Windows

As of R2019b Update 3, MATLAB on Windows ships with Perl version 5.30.1.

- See www.perl.org for a standard distribution of perl, perl source, and information about using perl.
- See https://metacpan.org/pod/HTML::Parser for a standard distribution of HTML::Parser, source code, and information about using HTML::Parser.
- See https://metacpan.org/pod/HTML::Tagset for a standard distribution of HTML:Tagset, source code, and information about using HTML:Tagset.

Compatibility Considerations

If you use the `perl` command on Windows platforms, see www.perl.org for information about using this version of the Perl programming language.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Microsoft Visual Studio® 2019 for C and C++	Windows
Discontinued	Intel Parallel Studio XE 2015 and XE 2016 for Fortran	Windows macOS

To ensure continued support for building your MEX files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Functionality being removed or changed

C++ interface treats read-only objects like C++

Behavior change

A C++ read-only object is an object declared with the C++ `const` attribute. You might get such an object as the output of a function or as a data member of a class. Starting in R2019b, the C++ interface for MATLAB honors the `const` type qualifier ensuring that the MATLAB behavior matches the C++ behavior of `const`-qualified types. MATLAB throws an error if you use a read-only object as follows:

- Passing the object to functions with non-const inputs
- Calling a non-const method on the object
- Modifying object properties

To test if an object is read-only, call the `clibIsReadOnly` function.

In R2019a, the interface ignores the `const` type qualifier, allowing the MATLAB user to utilize `const` objects incorrectly.

C++ interface returns type-specific empty values for nullptr

Behavior change

Starting in R2019b, the C++ interface returns type-specific `empty` values for functions that return `nullptr`. For more information about `nullptr` return types. see MATLAB to C++ Data Type Mapping.

- For type `double`, MATLAB continues to return `[]` for the value `double.empty`.
- For all other fundamental types, MATLAB returns an *`MLTYPE`*`.empty` value. To test for `nullptr` types, call the `isempty` function.
- For nonfundamental types, MATLAB returns a `nullptr` object. To test for `nullptr` objects, call the `clibIsNull` function.

In R2019a, for fundamental and nonfundamental types, the interface returns a `double.empty` (`[]`) value.

For example, suppose that these C++ functions return `nullptr`:

```
class A {
public:
    double val;
};

// Function returning nullptr object
A* returnNullptrObject() {
   return nullptr;
}

// Functions returning nullptr primitive type ptr
double* returnDoubleNullptr () {
    return nullptr;
}

const char* returnStringNullptr () {
    return nullptr;
}
```

R2019a	R2019b

For objects, MATLAB returns `[]` (`double.empty`).
 	
`nullReturn = clib.nullptr.returnNullptrObject`
 	
`nullReturn =`
`    []`
	MATLAB returns `nullPtr` for an object of class `A`.
	
	`nullReturn = clib.nullptr.returnNullptrObject`
	`nullReturn =`
	`    null A`

For fundamental types, MATLAB returns `[]`.
 	
`nullReturn = clib.nullptr.returnStringNullptr`
 	
`nullReturn =`
`    []`
	MATLAB returns empty `string` array for type `const char*`.
	`nullReturn = clib.nullptr.returnStringNullptr`
	
	`nullReturn =`
	`    0×0 empty string array`

For type `double`, MATLAB returns `[]`.
 	
`nullReturn = clib.nullptr.returnDoubleNullptr`
 	
`nullReturn =`
`    []`
	No change. MATLAB returns `[]`.
	
	`nullReturn = clib.nullptr.returnDoubleNullptr`
	
	`nullReturn =`
	`    []`

pyversion is not recommended
Still runs

`pyversion` is not recommended. Use `pyenv` instead. There are no plans to remove `pyversion` at this time.

To execute Python functions out of process, MATLAB provides a new function, `pyenv`. This function configures Python environment settings, including the version. Even if you do not use the out-of-process feature, MathWorks recommends using `pyenv` for managing Python settings. For more information, see Out-of-Process Execution of Python Functionality.

C MEX and engine applications: true, false, and bool defined by <stdbool.h>
Behavior change

The definition for `true`, `false`, and `bool` has changed for building MEX files and standalone MATLAB engine and MAT-file applications with C99 compatible compilers on Windows and Linux platforms. MATLAB defines these values using `<stdbool.h>` as defined by IEEE Std 1003.1:

```
The <stdbool.h> header shall define the following macros:
bool
    Expands to _Bool.
true
    Expands to the integer constant 1.
false
    Expands to the integer constant 0.
_bool_true_false_are_defined
    Expands to the integer constant 1.
```

In R2019a and earlier, MATLAB defined these values on Windows and Linux platforms as:

- `true` — `#defined` as `1`
- `false` — `#defined` as `0`
- `bool` — `typedef` as `unsigned char`

For Apple macOS platforms, there is no change.

actxcontrol, actxcontrollist, and actxcontrolselect functions will be removed in a future release
Warns

The `actxcontrol`, `actxcontrollist`, and `actxcontrolselect` functions will be removed in a future release. MATLAB will support COM server objects only.

R2019a

Environment

Live Editor Controls: Add check boxes, edit fields, and buttons to set variable values and run the live script

You can add check boxes and edit fields to your live scripts to interactively set variable values. You also can add a button to run the live script when clicked.

To add a check box, edit field, or button, go to the **Live Editor** tab, click  **Control** ▾, and select from the available controls. For more information, see Add Interactive Controls to a Live Script.

Live Editor Controls: Specify what code to run when a control value changes

By default, when you change the value of an interactive control, the Live Editor runs the section that contains the control. You can now configure an interactive control to run all sections, run the current section and all remaining sections, or to do nothing.

To configure the control, right-click the control and select **Configure Control**. Then, in the **Execution** section, select from the available options.

Configuring an interactive control to do nothing when changed is useful when your live script contains multiple interactive controls and you only want to run the code after changing all of their values. Add a button to the live script to run the code when clicked.

Live Editor Controls: Hide code when sharing and exporting live scripts with interactive controls

You can hide the code in a live script, showing only the interactive controls, output, and formatted text. Hiding the code is useful when sharing and exporting live scripts.

To hide the code in a live script, click the hide code  button to the right of the live script. To show the code again, click the output inline  button or the output on right  button.

If you export the live script to PDF, HTML, LaTeX, or Microsoft Word, the code remains hidden.

Live Editor Export: Save live scripts and functions as Microsoft Word documents

To create editable, static documents capable of being viewed outside of MATLAB, save live scripts and functions as Microsoft Word documents. To save a live script or function as a Microsoft Word document, on the **Live Editor** tab, select **Save** > **Export to Word**. This format is only available on Windows platforms.

For more information about sharing live scripts and functions, see Share Live Scripts and Functions.

Live Editor Output: Enable animations in plots to show changes in data over time

You can enable for-loop animations in the Live Editor to show changes in plotted data over time.

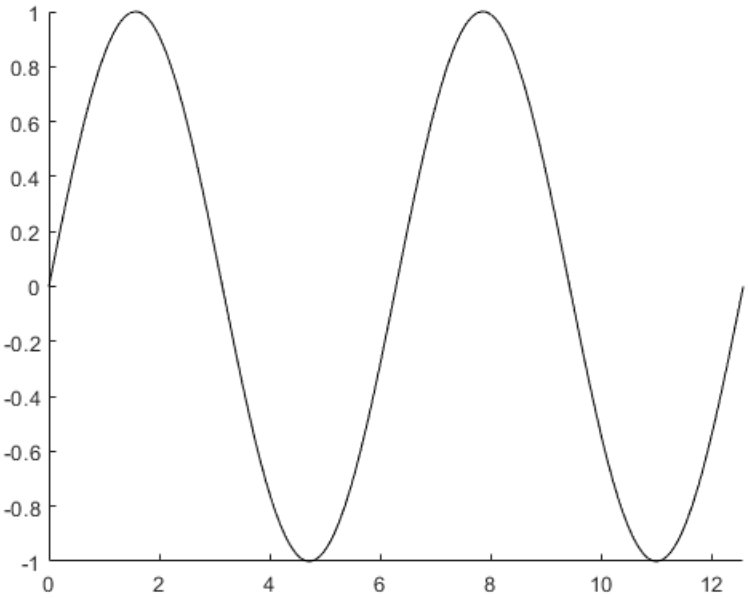
To enable animations in the Live Editor, set the `matlab.editor.AllowFigureAnimations` setting to `true`:

```
s = settings;
s.matlab.editor.AllowFigureAnimation.PersonalValue = true;
```

Note Enabling animations disables support for uicontrols in the Live Editor.

For example, this code turns on animations in the Live Editor, and then animates a line growing as it accumulates 2,000 data points. The `drawnow` function displays the changes after each iteration through the loop.

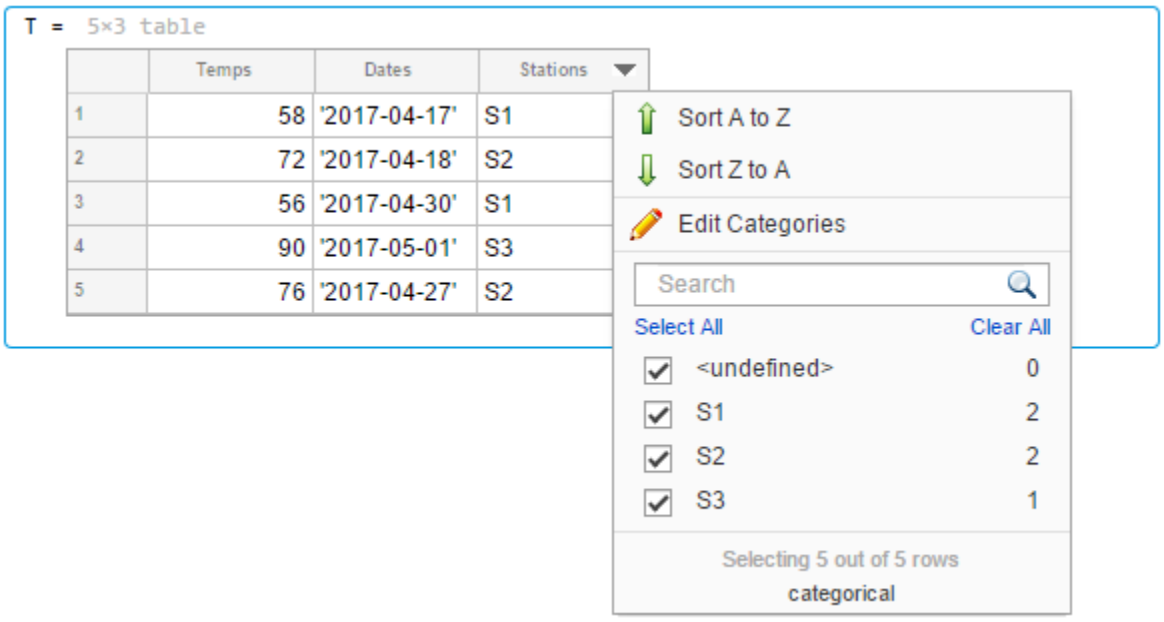
```
s = settings;
s.matlab.editor.AllowFigureAnimation.PersonalValue = true;

h = animatedline;
axis([0 4*pi -1 1])
x = linspace(0,4*pi,2000);

for k = 1:length(x)
    y = sin(x(k));
    addpoints(h,x(k),y);
    drawnow
end
```



Live Editor Output: Interactively clean categorical data and filter datetime and duration variables in table output

In the Live Editor, you can interactively clean categorical data and filter `datetime` and `duration` variables in table output.

To clean a categorical variable in a table, click the down arrow ▾ to the right of the variable name and select **Edit Categories**. Use the available options to create, remove, and merge categories.

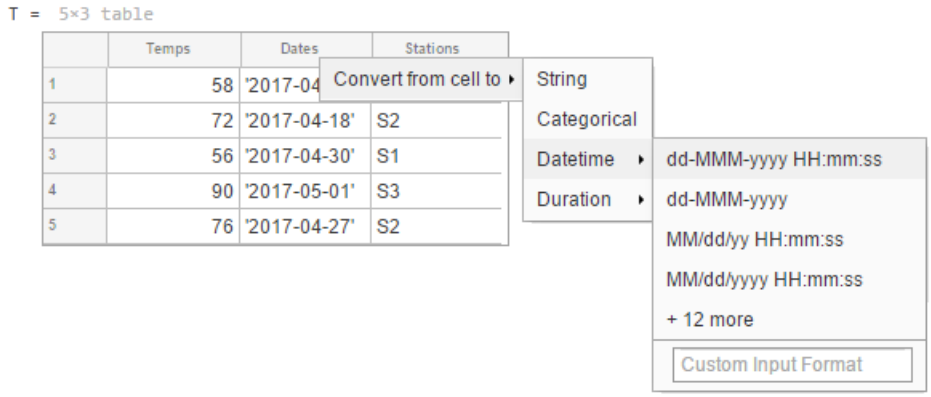


To filter a `datetime` or `duration` variable in a table, click the down arrow ▾ to the right of the variable name and select from the available filtering options.

To add the generated code to your live script, click the **Update Code** button below the table. Adding the generated code to your live script ensures that the cleaning and filtering is reproduced the next time you run the live script.

Live Editor Output: Interactively change the data type of variables in table output

In the Live Editor, you can interactively change the data type of a variable in table output. Right-click the variable column in the table, select **Convert from *`datatype`* to**, and select from the available options.



Live Editor Functions: Automatically convert selected code to a function

Break large scripts or functions into smaller pieces by converting selected code into functions in files or local functions. With one or more lines of code selected, on the **Live Editor** tab, in the **Code** section, click **Refactor**, and then select from the available options. MATLAB creates a function with the selected code and replaces the original code with a call to the newly created function.

MATLAB Online: Share folders and collaborate with others

Share your folders with a view-only link, or invite individual collaborators and set their editing permissions. Invitations can be accepted or declined.

After a folder is shared, you can manage the permissions of invited members, rescind invitations, or send additional invitations at any time.

Projects: Organize, manage, and share your work using projects

Create projects in MATLAB to organize and share your work with others. Use projects to find files required to run your code, manage and share files and settings, and interact with source control.

To create a project from an existing folder of files, in the desired folder, go to the **Home** tab, and select **New > Project > From Folder**. MATLAB creates the project and adds your existing files to the project.

MATLAB Startup: Execute MATLAB script or function non-interactively

To call a MATLAB script or function non-interactively, start MATLAB with the `-batch` option. The option is for non-interactive use in both scripting and command-line workflows. MathWorks recommends that you use the `-batch` option instead of the `-r` *`statement`* option for these cases.

For example, to run unit tests you created for your programs, from the operating system command prompt, type:

```
matlab -batch runtests
```

MATLAB:

- Starts without the desktop
- Does not display the splash screen
- Executes the `runtests` function
- Logs text to `stdout` and `stderr`
- Exits automatically with status

To test if a session of MATLAB is running in batch mode, call the `batchStartupOptionUsed` function.

For more information, see `matlab (Windows)`, `matlab (macOS)`, or `matlab (Linux)`.

Toolbox Packaging: Install required add-ons with custom toolboxes

When creating a custom toolbox, MATLAB detects the add-ons required by the toolbox. When someone installs your toolbox, the toolbox also downloads and installs the required add-ons.

Language and Programming

append Function: Combine strings

Combine text in string arrays, character vectors, and cell arrays of character vectors using the `append` function.

Unlike the `strcat` function, `append` treats all input data types the same. For example, if an input argument has trailing whitespace characters, then `append` always keeps them, even when the argument is a character vector.

MException class: Provide a suggested fix for an uncaught exception

Provide a suggested fix, using the `Correction` class, for an exception when it is thrown and not caught. Use the `addCorrection` method to add the correction to an `MException` object.

Functionality being removed or changed

Folders named resources are not allowed on the MATLAB path
Warns

Starting in R2019a, the `resources` folder is a reserved folder, and folders with the name `resources` are not allowed on the MATLAB path. In previous releases, these folders were allowed on the MATLAB path.

If a folder named `resources` is specified when calling the `addpath`, `userpath`, or `pathdef` functions, MATLAB returns a warning and the folder is not added to the path. If you have a folder named `resources`, MATLAB is unable to run any of the contents of that folder, even if the `resources` folder is the current folder.

Rename all folders on the path named `resources`, and move any files you want to run in MATLAB out of folders named `resources`.

Cell array expansion is consistent with general array expansion
Behavior change

Starting in R2019a, the dimensions of an expanded cell array are consistent whether you use curly braces or parentheses for indices. Previously, the output dimensions were different when you did not specify indices for all dimensions. Indexing with curly braces now matches the previous behavior for indexing with parentheses, which is consistent with general array expansion.

For more information, see the Compatibility Considerations section of `cell`.

Structure array expansion is consistent with general array expansion
Behavior change

Starting in R2019a, the dimensions of an expanded structure array are consistent whether you assign a value to a single field using dot notation or assign an entire structure to the array. Previously, the output dimensions were different when you did not specify indices for all dimensions. Assigning to a field using dot notation now matches the previous behavior of assigning a structure, which is consistent with general array expansion.

For more information, see the Compatibility Considerations section of `struct`.

Class properties using size validation no longer unconditionally reshape empty arrays
Behavior change

In previous releases, if a class defined a property using a size validation that contained unrestricted dimensions (indicated by a colon, such as `(:,:)`), then assigning an empty array of any size to the property resulted in an empty array of size `(0,0)`. For example, given this class definition:

```
classdef MyClass
    properties
        Prop1(:,:)
        Prop2
    end
end
```

Assigning an empty array of any dimension to `Prop1` always resulted in an empty array of dimensions `(0,0)`.

```
obj = MyClass;
obj.Prop1 = double.empty(0,5);
size(obj.Prop1)
```

```
ans =

     0     0
```

Assigning an empty array to `Prop2` produces the correct result because size validation with unrestricted dimensions is not used in the class.

```
obj = MyClass;
obj.Prop2 = double.empty(0,5);
size(obj.Prop2)
```

```
ans =

     0     5
```

Starting in R2019a, using unrestricted size validation for properties does not cause the size of empty arrays assigned to the properties to be reshaped to `(0,0)`. In R2019a, the same class definition produces these results for assignment to `Prop1`.

```
obj = MyClass;
obj.Prop1 = double.empty(0,5);
size(obj.Prop1)
```

```
ans =

     0     5
```

Defining classes and packages using schema.m will not be supported in a future release
Still runs

Support for classes and packages defined using `schema.m` files will be removed in a future release. Replace existing schema-based classes with classes defined using the `classdef` keyword.

First argument to ismethod must be an object
Behavior change in future release

The `ismethod` function is documented to look for a method of the object that is specified as the first input. However, the `ismethod` function treats `string` and `char` inputs as a class name and looks for the specified method in that class. Therefore, you cannot use `ismethod` to find a method of an input object that is a `string` or `char` array. In future releases, `ismethod` will return true only if the second input is the name of a method of the first input object. `ismethod` will not treat the first input as a class name.

For code that uses `ismethod` with a class name specified as a string scalar or character vector, you can substitute this expression as an alternative that will work in current and future versions.

```
any(strcmp('methodName', methods('ClassName')))
```

Program files larger than 128 MB or with high complexity will not be supported
Behavior change in future release

In a future release, running or opening program files larger than approximately 128MB will not be supported. For files that contain only code (for example, `.m` and `.p` files), this limit will affect the file size. For files that store more than just code (for example, `.mlx` files), it will affect the size of the code. Running statements larger than 128MB, either directly in the Command Window or using the `eval` function, also will not be supported. In addition, code with high levels of complexity, such as a large number of deeply nested `if` statements, will not be supported. Currently, these files and code are supported but can cause errors or unpredictable behavior.

Code that is too large or complex will not run or open in MATLAB, and MATLAB will display an error.

Large program file or statement sizes often occur when using large portions of code (for example, over 500 lines) to define variables with constant values. To decrease the size of these files, consider defining the variables and saving them in a data file (for example, a MAT-file or `.csv` file). Then you can load the variables instead of executing code to generate them. This not only decreases the file size of your program, but can also increase performance.

Data Analysis

xcorr and xcov Functions: Compute cross-correlation and cross-covariance in core MATLAB

You can now compute the cross-correlation and cross-covariance of data using MATLAB. Previously, `xcorr` and `xcov` were only available in the Signal Processing Toolbox™.

detrend Function: Remove piecewise polynomial trends, set continuity requirements, and specify sample points

The `detrend` function now offers additional functionality.

- In addition to the constant and linear methods for removing piecewise trends, you can specify higher degree polynomials. For example, `detrend(A,3)` removes a cubic trend from the data in `A`.
- When supplying break points, you can use the `'Continuous'` parameter to specify whether the fitted trend must be continuous.
- The `'SamplePoints'` parameter allows you to define the sample points associated with the input data.

groupcounts Function: Count the number of group elements for arrays, tables, and timetables

To count the number of elements in a group, use the `groupcounts` function.

grouptransform Function: Transform array data by group

In addition to tables and timetables, you can now transform data in an array by group using the `grouptransform` function.

filloutliers, isoutlier, and rmoutliers Functions: Detect outliers using percentiles

The `filloutliers`, `isoutlier`, and `rmoutliers` functions now offer Winsorization for detecting outliers using the `'percentiles'` option.

fillmissing and filloutliers Functions: Fill missing and outlier data using modified Akima interpolation

You can now fill missing and outlier data with modified Akima interpolation using the `'makima'` option in the `fillmissing` and `filloutliers` functions.

fillmissing Function: Specify missing value locations

To specify the locations of missing data when using the `fillmissing` function, use the `'MissingLocations'` parameter.

min and max Functions: Return index information when operating on more than one dimension and specify linear indices

When simultaneously operating on more than one dimension with the `min` and `max` functions, you can now return index information corresponding to the minimum and maximum values.

You can also return the linear indices corresponding to the minimum and maximum values of the input array using the `'linear'` option.

tall Arrays: Write custom sliding-window algorithms to operate on tall arrays

The functions `matlab.tall.movingWindow` and `matlab.tall.blockMovingWindow` enable you to write custom algorithms for sliding-window functions to operate on tall arrays.

tall Arrays: Operate on tall arrays with more functions, including groupcounts, intersect, and svd

The functions listed in this table now support tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

`addvars`	`movevars`
`cospi`	`normalize`
`groupcounts`	`removevars`
`grouptransform`	`sinpi`
`inner2outer`	`splitvars`
`intersect`	`svd`
`mergevars`	`union`

In addition, some functions have removed limitations with tall arrays.

Functions	Added Support
`stack`	The two-output syntax `[S,iu] = stack(___)` now supports tall arrays. The second output returns indices that describe the mapping of rows in the stacking operation. *Previously, multiple outputs were not supported for tall arrays.*
`groupsummary`	Grouped calculations on tall matrices and arrays are now supported. *Previously, the first input was required to be a tall table or tall timetable.*

Functionality Being Removed or Changed

Default random number generator change for tallrng

Behavior change

Starting in R2019a, the default random number generator for `tallrng` is `threefry`. This generator offers performance enhancements for parallel calculations over the old default. In releases up to R2018b, the default random number generator for `tallrng` was `combRecursive`.

With a different default generator, MATLAB will generate different sequences of random numbers by default in the context of tall arrays. However, the statistics of these calculations will remain unaffected. Therefore, you should update any code that relies on the *specific* random numbers being generated. However, most calculations on the random numbers should be unaffected.

To set the generator to the settings used by default in R2018b and earlier releases, use the command:

```
tallrng(0,'combRecursive')
```

Data Import and Export

readmatrix, readvars, and readcell Functions: Read tabular data as a matrix, variables, or a cell array

Read column-oriented data from text or spreadsheet files into a matrix, variables, or a cell array.

- `readmatrix` — Read homogeneous column-oriented data into a matrix.
- `readvars` — Read column-oriented data into variables. Each variable corresponds to a column of data in the file.
- `readcell` — Read heterogeneous data into a cell array.

writematrix and writecell functions: Write tabular data from a matrix or cell array to a text or spreadsheet file

Write data from a matrix or a cell array to a text or spreadsheet file.

- `writematrix` — Write a homogeneous array to a file.
- `writecell` — Write a cell array to a file.

readtimetable and writetimetable Functions: Read and write timetables

Read and write timetables in MATLAB.

- Use the `readtimetable` function to read timetables from text or spreadsheet files.
- Use the `writetimetable` function to write timetables to text or spreadsheet files.

detectImportOptions Function: Improve detection of import options for text and spreadsheet files

Improve the detection of import options for text and spreadsheet files by passing additional information to the `detectImportOptions` function using these name-value pairs.

- `'ThousandsSeparator'` — Character separating thousands groups (numeric variables only)
- `'DecimalSeparator'` — Character separating integer part from fractional part (numeric variables only)
- `'TrimNonNumeric'` — Remove non-numeric characters from numeric variables (numeric variables only)
- `'ConsecutiveDelimitersRule'` — Procedure to handle consecutive delimiters (text files only)
- `'LeadingDelimitersRule'` — Procedure to handle leading delimiters (text files only)
- `'TreatAsMissing'` — Text to interpret as missing data (text files only)
- `'ReadRowNames'` — Read first column as row names
- `'ReadVariableNames'` — Read first row as variable names

For more information, see the `setvaropts` and `detectImportOptions` reference pages.

parquetread, parquetwrite, and parquetinfo Functions: Read, write, and get information from Parquet files

Import and export column-oriented data from Parquet files in MATLAB. Parquet is a columnar storage format that supports efficient compression and encoding schemes. To work with the Parquet file format, use these functions.

- `parquetread` — Read columnar data from a Parquet file.
- `parquetwrite` — Write columnar data to a Parquet file.
- `parquetinfo` — Get information about a Parquet file.

For more information on the Parquet file format, see https://parquet.apache.org/.

write Function: Write tall arrays to Parquet files

The `write` function now supports writing tall arrays to Parquet files. To write a tall array, set the `FileType` parameter to `'parquet'`, for example:

```
write('C:\myData',tX,'FileType','parquet')
```

Import Tool: Generate improved code when importing from text files

Import Tool now functions consistently across different platforms and generates code that is easy to read for importing text files. For more information, see Import Text File Data Using Import Tool.

thingSpeakRead and thingSpeakWrite Functions: Read or write data to the ThingSpeak IoT platform

Access IoT data in ThingSpeak™ channels:

- Use `thingSpeakRead` to read data from ThingSpeak channels.
- Use `thingSpeakWrite` to write data to ThingSpeak channels.

For more information on the ThingSpeak platform, see https://thingspeak.com/.

writetable and imwrite Functions: Write to web-based storage services like Amazon Web Services and Azure Blob Storage

Write tabular data and image files to remote locations using the `writetable` and `imwrite` functions. When writing data to remote locations, you must specify the full path using a uniform resource locator (URL). For example, write a `csv` file and a `jpg` file to Amazon S3 Cloud:

```
writetable(T,'s3://bucketname/path_to_file/my_text_file.csv');
imwrite(I,'s3://bucketname/path_to_file/my_image.jpg');
```

For more information on setting up MATLAB to access your online storage service, see Work with Remote Data.

ParquetDatastore Object: Create a datastore for a collection of Parquet files

Read a collection of Parquet files into MATLAB workspace using `parquetDatastore`.

For more information on the Parquet file format, see https://parquet.apache.org/.

ImageDatastore Object: Create a subset of an existing datastore

Create a subset of an image datastore using the `subset` method.

DsFileSet Object: Create a subset of a file collection

You can create a subset of a `DsFileSet` object by using the `subset` method. The `DsFileSet` object helps you manage the iterative processing of large collections of files.

FileDatastore Object: Read large files by importing the file in smaller portions

Read and process large files in smaller portions. For example, you can create a datastore that reads one array at a time from a large MAT-file that does not fit in the available memory. To set up your datastore to perform partial reads, use these name-value pairs: `'ReadMode'`, `'PreviewFcn'`, and `'BlockSize'`.

For more information, see `fileDatastore`.

Datastores: Combine and transform datastores

Perform `combine` and `transform` operations on existing datastores.

- `combine` — Combine two or more datastores and return a new datastore representing the underlying datastores.
- `transform` — Transform an input datastore by using a specified transformation function and return the transformed datastore.

Custom Datastore: Read Hadoop based data from files, databases, and other non-file-based locations

Author a custom datastore to access data stored in files or non-file-based data sources such as a databases using `matlab.io.datastore.HadoopLocationBased` mixin. Use this extension to specify the location of your data in Hadoop®. A custom datastore with the `HadoopLocationBased` mixin makes computations more efficient by leveraging the location of the data. With your custom datastore you can perform big data analysis by using `tall` arrays and `mapreduce`.

For more information on the custom datastore framework, see Develop Custom Datastore.

VideoReader function: Generate C and C++ code

The `VideoReader` function supports C and C++ code generation using MATLAB Coder™.

ind2rgb function: Generate C and C++ code

The `ind2rgb` function supports C and C++ code generation using MATLAB Coder.

Scientific File Format Libraries: NetCDF Library upgraded to version 4.6.1

The NetCDF library is upgraded to version `4.6.1`.

web function: Open external sites in system browser instead of MATLAB browser

You can change the default behavior of the `web` function to open external sites in your system browser instead of the MATLAB browser. Using the system browser is recommended when opening external sites. To change the default behavior, go to the **Home** tab, and in the **Environment** section, click **Preferences**. Select **MATLAB > Web**, and in the **System Web browser** section, select **Use system web browser when opening links to external sites (recommended)**.

Functionality being removed or changed

NumberOfChannels property of the audioplayer and audiorecorder Objects is not recommended

Still runs

The `NumberOfChannels` property of the `audioplayer` and `audiorecorder` objects is not recommended. Use the name `NumChannels` instead. To update your code, change instances of `NumberofChannels` to `NumChannels`. The values of the properties are the same. There are no plans to remove the `NumberOfChannels` property at this time.

web Function

Behavior change in future release

In future releases, the `web` function will open external sites using your system browser by default. Currently, the `web` function opens external sites using the MATLAB browser. Using the system browser is recommended when opening external sites.

To change the default browser, go to the **Home** tab, and in the **Environment** section, click **Preferences**. Select **MATLAB > Web** and in the **System Web browser** section, select **Use system web browser when opening links to external sites (recommended)**.

hdftool is not recommended

Still runs

In a future release, `hdftool` will be removed. To import HDF4 or HDF-EOS files, use the `hdfread` function instead.

csvread and csvwrite functions are not recommended

Still runs

`csvread` and `csvwrite` are not recommended. Use `readmatrix` and `writematrix` instead. There are no plans to remove `csvread` and `csvwrite`.

This table shows typical usages of `csvread` and `csvwrite` and how to update your code to use `readmatrix` and `writematrix` instead.

Not Recommended	Recommended
`M = csvread(filename)`	`M = readmatrix(filename)`
`csvwrite('mydata.txt',M)`	`writematrix(M,'mydata.txt')`

For more information, see `readmatrix` and `writematrix`.

dlmread and dlmwrite functions are not recommended
Still runs

`dlmread` and `dlmwrite` are not recommended. Use `readmatrix` and `writematrix` instead. There are no plans to remove `dlmread` and `dlmwrite`.

This table shows typical usages of `dlmread` and `dlmwrite` and how to update your code to use `readmatrix` and `writematrix` instead.

Not Recommended	Recommended
`M = dlmread(filename)`	`M = readmatrix(filename)`
`dlmwrite('mydata.txt',M)`	`writematrix(M,'mydata.txt')`

For more information, see `readmatrix` and `writematrix`.

xlsread and xlswrite functions are not recommended
Still runs

`xlsread` and `xlswrite` are not recommended. Instead of `xlsread` and `xlswrite`:

- Use `readtable` and `writetable` for reading and writing mixed numeric and text data.
- Use `readmatrix` and `writematrix` for reading and writing homogeneous text or numeric data.
- Use `readcell` and `writecell` for reading and writing mixed numeric and text data.

There are no plans to remove `xlsread` and `xlswrite`.

This table shows typical usages of `xlsread` and `xlswrite` and how to update your code to use the recommended read and write functions.

Not Recommended	Recommended

Read spreadsheet data as a matrix using `xlsread`:
 	
`M = xlsread(filename)`
	Read spreadsheet data as a table:
	
	`T = readtable(filename)`
	
	However, to continue reading your data as a matrix, use:
	
	`M = readmatrix(filename)`

Not Recommended	Recommended

Read spreadsheet data as a cell array using `xlsread`:
 	
`[~,~,C] = xlsread(filename)`
	Import spreadsheet data as a table:
	
	`T = readtable(filename)`
	
	However, to continue importing your data as a cell arrary, use:
	
	`C = readcell(filename)`

Read a specific sheet and range as a matrix using `xlsread`:
 	
`M = xlsread(filename,sheet,range)`
	Read a specific sheet and range as a table:
	
	`T = readtable(filename,'Sheet',sheet,'Range',range)`
	
	However, to continue reading your data as a matrix, use:
	
	`M = readmatrix(filename,'Sheet',sheet,'Range',range)`

Read a specific sheet and range as a cell array using `xlsread`:
 	
`[~,~,C] = xlsread(filename,sheet,range)`
	Read a specific sheet and range as a table:
	
	`T = readtable(filename,'Sheet',sheet,'Range',range)`
	
	However, to continue reading your data as a cell array:
	
	`C = readcell(filename,'Sheet',sheet,'Range',range)`

Write tabular data to spreadsheets using `xlswrite`:
 	
`xlswrite(filename,M)`
	To write tabular data to spreadsheets, use one of these options instead.
	
	Write a table:
	
	`writetable(T,filename)`
	
	Write a matrix:
	
	`writematrix(M,filename)`
	
	Write a cell array:
	
	`writecell(C,filename)`

For more information, see `readmatrix`, `writematrix`, `readcell`, `writecell`, `readtable`, and `writetable`.

`HadoopFileBased` is not recommended
Still runs

`HadoopFileBased` is not recommended. Use `HadoopLocationBased` instead. There are no plans to remove `HadoopFileBased`.

Starting in R2019a, use the `HadoopLocationBased` mixin to add Hadoop support to your custom datastore. The `HadoopLocationBased` mixin provides support for non-file-based data where as `HadoopFileBased` supports file-based data only.

For more information on the custom datastore framework, see Develop Custom Datastore.

Mathematics

Solve assignment problem with matchpairs and equilibrate

New functions enable you to solve the assignment problem in a variety of contexts.

- `matchpairs` — Create a linear mapping between the rows and columns of a cost matrix. This assigns rows to columns in such a way that the global cost is minimized.
- `equilibrate` — Permute and rescale a matrix `A` such that the new matrix `B = R*P*A*C` has only 1s and -1s on its diagonal, and all off-diagonal entries are not greater than 1 in magnitude. When computing a preconditioner to iteratively solve a linear system, use equilibration to improve the condition of a matrix and allow for improved preconditioners.

graph and digraph Objects: Construct graphs with categorical nodes

The `graph`, `digraph`, and `addedge` functions now support categorical node names as inputs. This enables you to use data that is imported as `categorical` to create a graph, without the need for data type manipulation.

Graphics

parallelplot Function: Visualize tabular or matrix data with multiple columns by using a parallel coordinates plot

To create a parallel coordinates plot, use the `parallelplot` function. Rows of the input data correspond to lines in the plot, and columns of the input data correspond to coordinates in the plot. To group the lines in the plot, you can use either the `'GroupVariable'` name-value pair argument with tabular data or the `'GroupData'` name-value pair argument with matrix data.

Data Tips: Pin and customize data tips in charts

The data tips that appear as you hover over a chart become persistent (pinned) when you click them. Clicking a second time unpins the data tip.

For some types of charts, you can customize the contents of the data tips. For example, you can edit the data tip labels, change the displayed values, or change the font size. Also, you can add or delete rows from the data tips. Charts that support these customizations include `Scatter`, `Stair`, `Stem`, `Line`, and `Surface` objects with a `DataTipTemplate` property.

- To edit the labels interactively, double-click a label, type the text you want, and then click outside the data tip. To make other customizations interactively, right-click the data tip and select **Edit Properties...**. Use the fields in the Property Inspector that opens to make any changes.
- To customize the data tip programmatically, use the `DataTipTemplate` property of the chart object. For example, this code plots sample patient data from a table as a scatter chart. Then it changes the font size and labels of the data tips. For more information, see `DataTipTemplate`.

  ```
  tbl = readtable('patients.xls');
  s = scatter(tbl.Weight,tbl.Height);
  s.DataTipTemplate.FontSize = 12;
  s.DataTipTemplate.DataTipRows(1).Label = 'Weight';
  s.DataTipTemplate.DataTipRows(2).Label = 'Height';
  ```

  You can add a new row to the data tip using the `dataTipTextRow` function. For example, add a third row that shows the patient name from the table.

  ```
  s.DataTipTemplate.DataTipRows(3) = dataTipTextRow('Name',tbl.LastName);
  ```

Axes Interactions: Customize chart interactions such as dragging to pan or scrolling to zoom

Create a customized set of chart interactions by setting the `Interactions` property of the axes. These interactions are built into the axes and are available without having to select any buttons in the axes toolbar. Some types of interactions are enabled by default, depending on the content of the axes.

For more information, see Control Chart Interactivity.

Ruler Panning: Pan an axis to change its limits without having to use the pan tool

Drag an axis to change the limits along a dimension of a plot. This functionality is available for most Cartesian plots, even when the pan tool in the axes toolbar is disabled.

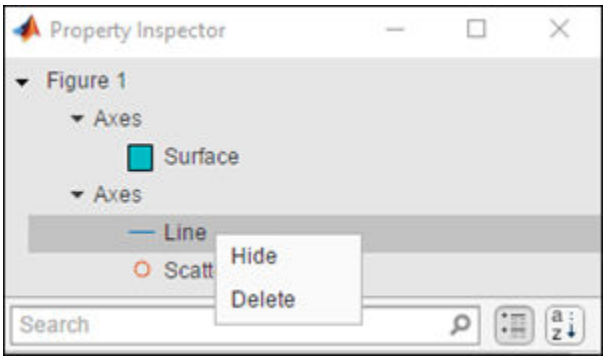
Property Inspector: Navigate and control visibility of graphics objects interactively

You can use the object browser that appears at the top of the **Property Inspector** to navigate and control the visibility of graphics objects. When you select an object using the object browser, the object appears selected in the figure and the properties appear in the inspector.

The object browser has a collapsed view and an expanded view.

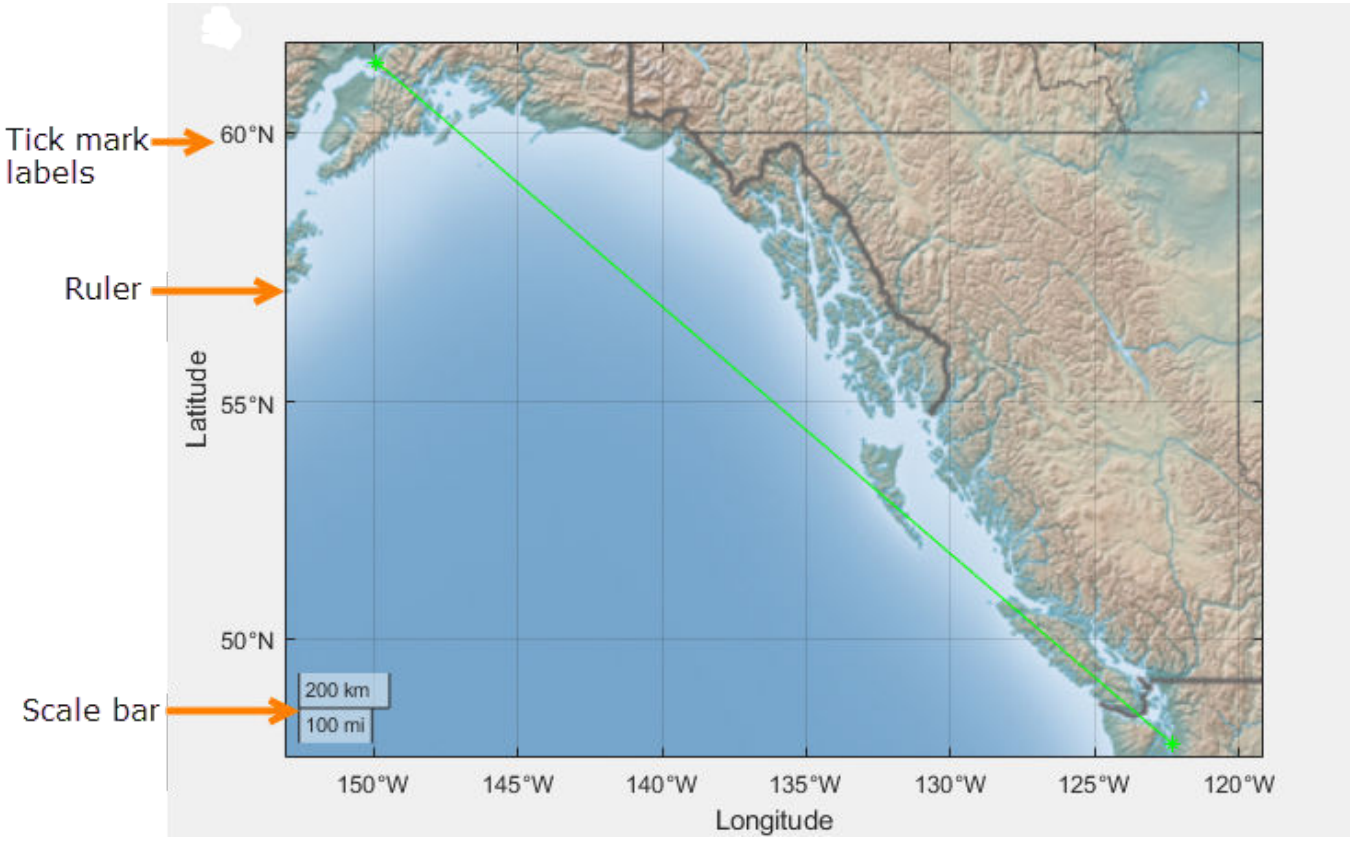
- The collapsed view (default view) shows the currently selected object and its direct hierarchy. Click one of the object names to see its properties in the Property Inspector.

- The expanded view shows the graphics object hierarchy of the figure. Right-click an object name to show, hide, or delete the graphics object. Select multiple objects using **Ctrl**+click.



Geographic Plots: Geographic rulers, scale bar, CurrentPoint, and ginput

Plots on geographic axes include customizable rulers and a scale bar.



Use the `geotickformat` function to customize rulers.

`GeographicAxes` support the `CurrentPoint` property. Use this property to get the current coordinates of the mouse pointer on a geographic axes.

Graphics Export: Export axes with tighter cropping using the axes toolbar

Click or tap the export button in the axes toolbar to save the axes as an image or PDF file. The saved content is tightly cropped around the axes with minimal white space.

Chart Resizing: Resize charts with improved layouts

The layout is improved when you resize a chart that can be a child of a figure (such as a heatmap). This automatic resizing behavior adjusts the font sizes and spacing between elements in the chart to provide the best possible presentation for the new size.

Changing the `FontSize` property of a chart disables the automatic resizing of the fonts.

Colors Values: Specify colors using hexadecimal color codes

Specify hexadecimal color codes when setting the color of graphics objects. For example, `set(gca,'XColor','#FF8800')` sets the *x*-axis color to orange. Use either the six-digit or three-digit form to specify a color. The characters are not case-sensitive. Thus, `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Categorical Values: Specify categorical arrays for functions and objects that use lists of text

Many functions and object properties that use lists of text items now accept categorical arrays in addition to cell arrays and string arrays. When you specify a categorical array, MATLAB uses the values in the array, not the categories. Thus you might need to write additional code to extract the values you want to use. For example, this code finds the unique entries of the `cities` categorical array before setting the *x*-axis tick labels.

```
bar([10 20 30])
cities = categorical({'Boston','Boston','Berlin','Paris','Berlin'});
xticklabels(unique(cities))
```

See the documentation for a specific function or object to determine whether it accepts categorical values.

rendererinfo Function: Get renderer information for any axes

Use the `rendererinfo` function to get information about the graphics renderer used for any axes or chart that can be a child of a figure.

Compatibility Considerations

Use the `rendererinfo` function instead of the `opengl` function to get information about the graphics renderer.

Functionality being removed or changed

Using the opengl function to get information about the graphics renderer is not recommended

Still runs

Using the `opengl` function to get information about the graphics renderer is not recommended. Specifically, these syntaxes are not recommended:

- `opengl info`
- `d = opengl('data')`

There are no plans to remove support for these syntaxes at this time. Instead of calling `opengl` to get the renderer information, call the `rendererinfo` function instead:

```
info = rendererinfo(ax)
```

Specify `ax` as any type of axes or a chart that can be a child of a figure (such as a heatmap). The output is a structure containing most of the same information as the `opengl` function provides.

Fields in opengl Structure	Corresponding Fields in rendererinfo Structure
`d.Version`	`info.Version`
`d.Vendor`	`info.Vendor`
`d.Renderer`	`info.RendererDevice`
`d.RendererDriverVersion`	`info.Details.RendererDriverVersion`
`d.RendererDriverReleaseDate`	`info.Details.RendererDriverReleaseDate`
`d.MaxTextureSize`	`info.Details.MaxTextureSize`
`d.Visual`	No longer needed
`d.Software`	This information is stored in `info.GraphicsRenderer`, but to get the equivalent logical value, use `strcmp(info.GraphicsRenderer,'OpenGL Software')`
`d.HardwareSupportLevel`	`info.Details.HardwareSupportLevel`
`d.SupportsGraphicsSmoothing`	`info.Details.SupportsGraphicsSmoothing`
`d.SupportsDepthPeelTransparency`	`info.Details.SupportsDepthPeelTransparency`
`d.SupportsAlignVertexCenters`	`info.Details.SupportsAlignVertexCenters`
`d.Extensions`	No longer needed
`d.MaxFrameBufferSize`	`info.Details.MaxFrameBufferSize`

Heatmaps interpret text using TeX markup
Behavior change

Starting in R2019a, heatmaps created with the `heatmap` function interpret text using TeX markup instead of displaying the literal characters. If you want to use a TeX markup character in regular text, such as an underscore (_), then insert a backslash (\) before the character you want to include. The backslash is the TeX escape character. For more information on using TeX markup, see the Interpreter property of the text object.

App Building

uiimage Function: Display an icon, logo, or picture in apps and on the App Designer canvas

To display a picture, icon, or logo in your app, call the `uiimage` function programmatically or, in App Designer, drag and drop an image component from the **Component Library** onto the canvas.

Image components are supported only in App Designer apps and in figures created with the `uifigure` function.

uitable Function: Sort tables interactively when using table arrays

To create tables that can be sorted interactively, use `table` arrays and configure the `ColumnSortable` property of the `Table` object. Use the `DisplayData` property and a `DisplayDataChangedFcn` callback if you want to update your visualizations based on how a user sorts a table containing `table` data.

Auto Resize: Automatically resize components when an app is made smaller

When a parent container is resized smaller than its initial size, `AutoResizeChildren` now reduces the white space between components and shrinks the components themselves to maintain usability. For more information, see Managing Resizable Apps in App Designer.

Scrolling Grids: Create apps with scrollable grids

Enable interactive scrolling in your grid layout manager by setting the `Scrollable` property of the grid to `'on'`. See `uigridlayout` for more information.

App Designer: Create apps that automatically reflow content based on device size

Create 2-panel or 3-panel preconfigured apps that automatically resize and reflow content based on screen size, screen orientation, and platform. Use apps with auto-reflow if you expect to run or share your apps across multiple environments or desktop resolutions. For more information, see Apps with Auto-Reflow.

App Designer: Add and configure a grid layout manager on the App Designer canvas

Structure the layout of your app by dragging a grid layout manager from the **Component Library** onto the canvas. To configure the grid layout in **Design View**, select the  icon from the upper-left hand corner of the grid, or right-click and select **Configure grid layout**. Then, select a row or column to edit. For more information, see the `uigridlayout` function or `GridLayout Properties`.

Grid layout managers are supported only in App Designer apps or in figures created with the `uifigure` function.

App Designer: Rearrange the order of callbacks

To rearrange the order of callbacks, go to the **Code Browser**, select the callback you wish to move, and then drag the callback into a new position in the list. This also repositions the callback in the editor.

App Designer: Create new apps using App Designer Start Page options

From the App Designer Start Page you can now do the following.

- Create a new blank app or a new responsive app with auto-reflow.
- Start a tutorial or access featured content from the top banner.
- View a list of your recent apps.
- Open apps from a file path.
- Select an example from the **Getting Started** or **Programming Tasks** example sections.

The Start Page appears when you launch App Designer. Once you are in the design environment, you can get back to the Start Page by selecting the New  icon from the **Designer** toolstrip.

App Designer: Control font, code, and autosave settings using MATLAB Preferences

Control these settings through the Preferences dialog box

- Autocoding for patterns like parentheses, block endings, and comment wrapping
- Keyboard preferences for automatic code suggestions and completions
- Autosave preferences upon clicking away from a file
- Font size preference for App Designer **Code View**

Changes to the autosave and font size preferences apply to only the App Designer Editor. When you set the autocoding or keyboard preferences, the change applies to the MATLAB Editor and to App Designer.

App Designer: Access context-sensitive help in Code View

To open the documentation for a component, function, or callback in your code, highlight the element and then press **F1**, or right-click and choose help for the code element you selected.

App Designer: Zoom in App Designer

Hold **Ctrl** and move the scroll wheel in the App Designer window to zoom in or out. To return to the default scale, press **Ctrl+0**.

Graphics Support: Explore data using axes toolbar and data tips in apps created with the uifigure function

Use the axes toolbar and data tips to explore plotted data interactively in App Designer apps and in figures created with the `uifigure` function. The axes toolbar and data tips are on by default for `axes` and `uiaxes` objects in a `uifigure`.

Compatibility Considerations

- Axes toolbar — In previous releases, the axes toolbar was not enabled for `axes` or `uiaxes` objects in a `uifigure` object. Now, it is enabled by default. You can turn it off by setting the `Visible` property of the `AxesToolbar` object to `'off'`. For more information, see `AxesToolbar Properties`.
- Data tips — In previous releases, data tips were not enabled for `axes` or `uiaxes` objects in a `uifigure` object. Now, to control whether the axes interactions are enabled, use the `disableDefaultInteractivity` and `enableDefaultInteractivity` functions. For example,

```
uf = uifigure;
ax = axes(uf);
plot(ax,rand(5))
disableDefaultInteractivity(ax)
```

Deployed Web Apps: Share resizeable apps or create apps that open web pages

In deployed web apps you can now do the following:

- Interactively resize your web app.
- Program your web app to open another URL using the `web` function.

For information about other new features of deployed web apps, see Release Notes (MATLAB Compiler).

MATLAB Online: Create and edit App Designer apps using MATLAB Online

Create or edit apps in MATLAB Online using the App Designer development environment (supported only for Google Chrome browsers).

App Testing Framework: Perform hover gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.hover` method enables you to perform hover gestures in tests on axes, UI axes, and UI figure objects. For more information, see the `hover` reference page.

App Testing Framework: Perform press gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.press` method enables you to perform press gestures in tests on UI axes and UI figure objects. For more information, see the `press` reference page.

App Testing Framework: Perform type gesture on date picker objects

The `matlab.uitest.TestCase.type` method enables you to perform type gestures on date picker objects. For more information, see the `type` reference page.

Functionality Being Removed or Changed

`javacomponent` function and `JavaFrame` property will be removed in a future release
Warns

The undocumented `javacomponent` function and `JavaFrame` property will be removed in a future release. The `JavaFrame` property still runs, but returns a warning. The `javacomponent` function still runs, without warning, but will begin to warn in an upcoming release. Users are encouraged to update their code to use documented alternatives. For a list of documented functionality you can use instead, see Java Swing Alternatives for MATLAB Apps on mathworks.com.

Support for running deployed web apps in Internet Explorer will be removed in a future release
Still runs

Support for running deployed web apps in Internet Explorer will be removed in a future release. Use the current versions of Google Chrome (recommended), Safari, Firefox, or Microsoft Edge to run deployed web apps instead.

Performance

MATLAB and Simulink startup on macOS platforms

To prevent performance regression at startup for MATLAB and Simulink on macOS platforms, MathWorks recommends using MATLAB R2018b Update 4 or later with macOS 10.13.6 Update or later.

sortrows Function: Sort rows of large matrices faster

For large matrices, you can now sort rows faster using the `sortrows` function.

For example, on a test system, this code runs faster in R2019a than in previous releases.

```
A = repmat(eye(200),500,1);
sortrows(A)
```

uitable Function: Faster performance using table arrays

Tables created with the `uitable` function and with data specified as a `table` array have better rendering performance and higher frame rates while scrolling. Tables that use `table` arrays render up to 40% faster, and interaction performance (like scrolling) is up to 75% faster. For example, on a test system, this code renders the tables faster in R2019a than in previous releases.

```
rows = 10000;
columns = 25;
data = array2table(randi(30, [rows, columns]));
fig = uifigure;
tbl = uitable(fig, 'Data', data);
```

For more information about using `table` arrays in `Table` UI components, see Table Array Data Types in App Designer Apps.

Software Development Tools

checkcode Function: Get the modified cyclomatic complexity of functions

Use the `checkcode` function with the `'modcyc'` option to get the modified cyclomatic complexity of each function in a file. The modified cyclomatic complexity for a function is equal to the McCabe complexity except for one difference. McCabe complexity counts each individual case within a `switch` statement as 1, while modified cyclomatic complexity counts the entire `switch` statement as 1. In general, `switch` statements are simpler than nested `if-elseif-else` statements and therefore, the modified cyclomatic complexity is often considered a better measure of code complexity.

Source Control Integration: Synchronise MATLAB Git status with external Git clients

If you use an external Git client, MATLAB now listens to external changes to working copies of `.git` folders and refreshes the file status if needed. MATLAB keeps the Git file status in sync when using another Git client, both in the MATLAB current folder and in a project.

Unit Testing Framework: Display code coverage metrics in HTML format

The `matlab.unittest.plugins.codecoverage.CoverageReport` class provides an HTML code coverage report format to display code coverage metrics. Use this format with `matlab.unittest.plugins.CodeCoveragePlugin` to produce the report.

Unit Testing Framework: Specify sources for collections of code coverage data with runtests

The `runtests` function enables you to specify the source code files to include in the code coverage report. Use the `ReportCoveragefor` name-value input to specify the files or folders containing source files to include in the tests.

Unit Testing Framework: runperf collects more samples to achieve its target margin of error

The default maximum number of sample measurements that `runperf` makes when running performance measurements has increased to 256. Specify the number of sample measurements using the `matlab.perftest.TimeExperiment.limitingSamplingError` method.

Unit Testing Framework: Return performance test results as TimeResult arrays

The `runperf` function now returns a `matlab.perftest.TimeResult` array containing the results of the specified performance tests. This class derives from the `matlab.unittest.measurement.MeasurementResult` class, which is now an abstract class.

Unit Testing Framework: Load previously saved MeasurementResult objects as DefaultMeasurementResult

`MeasurementResult` objects saved in previous releases are loaded as `matlab.unittest.measurement.DefaultMeasurementResult` objects. This class is derived from the `MeasurementResult` class.

Unit Testing Framework: Use matlab.unittest.fixtures.Fixture.onFailure method only in subclasses

The `onFailure` method now has `protected` access. In previous releases, `onFailure` had `public` access. This change better supports the use of `onFailure` to produce additional diagnostics in case of a failure during fixture setup or teardown in classes derived from `Fixture`.

Unit Testing Framework: Compare tables that contain no rows

In previous releases, for tables that had no rows (that is, that had a first size dimension of zero), the `matlab.unittest.constraints.IsEqualTo` constraint did not compare the table column variables when determining equality. Now, `IsEqualTo` always compares the size and type of each column variable.

For example, comparing these two tables fails now because the column variables are different types (`double` and `cell`).

```
tc = matlab.unittest.TestCase.forInteractiveUse;
a = table(zeros(0,2));
b = table({});
tc.verifyEqual(a,b)
```

```
Verification failed.
    ----------------------
    Framework Diagnostic:
    ----------------------
    verifyEqual failed.
    --> Path to failure: <Value>.Var1
        --> Classes do not match.

            Actual Class:
                double
            Expected Class:
                cell

        Actual double:
          0×2 empty double matrix
        Expected cell:
          0×0 empty cell array

    Actual Value:
      0×1 empty table
    Expected Value:
      0×1 empty table
```

Unit Testing Framework: Create test suite array from tests in project

The `fromProject` method enables you to create a test suite array from the files in a project that are labeled with the Test classification. For more information, see the `matlab.unittest.TestSuite.fromProject` reference page.

Unit Testing Framework: Run tests from files in project using runtests or testsuite

Run test files from projects using the `runtests` or `testsuite` functions. The `IncludeReferenceProjects` name-value pair argument enables you to include in the test suite files from a project that are labeled with the Test classification.

Unit Testing Framework: Specify verbosity enumeration as a string or character vector

You can specify the verbosity level argument for the following methods as a string scalar or character vector that correspond to the `matlab.unittest.Verbosity` enumeration member name.

- `matlab.unittest.TestCase.log`
- `matlab.unittest.TestRunner.withTextOutput`
- `matlab.unittest.plugins.TestRunProgressPlugin`
- `matlab.unittest.plugins.LoggingPlugin.withVerbosity`
- `matlab.unittest.plugins.DiagnosticsOutputPlugin`

App Testing Framework: Perform hover gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.hover` method enables you to perform hover gestures in tests on axes, UI axes, and UI figure objects. For more information, see `hover`.

App Testing Framework: Perform press gesture on axes, UI axes, and UI figures

The `matlab.uitest.TestCase.press` method enables you to perform press gestures in tests on UI axes and UI figure objects. For more information, see `press`.

App Testing Framework: Perform type gesture on date picker objects

The `matlab.uitest.TestCase.type` method enables you to perform type gestures on date picker objects. For more information, see `type`.

Mocking Framework: Create mocks for classes that use custom metaclasses

The unit testing framework can now create mocks for classes that use custom metaclasses to define custom class, property, method, and event attributes.

Mocking Framework: Create mocks for classes that use property validation

The unit testing framework can now create mocks for classes that use property validation. For information on property validation, see Validate Property Values.

Mocking Framework: Specify which methods to mock

When creating a mock object, you can control which methods are mocked in the test case. Use the `createMock` method with the `MockMethods` name-value pair argument to specify the method to mock. This feature enables tests to mock only those methods that are important to the test case, which can improve performance when superclasses define many methods.

Functionality being removed or changed

matlab.unittest.fixtures.Fixture.onFailure method has protected access
Behavior change

In release R2019a, the `onFailure` method `Access` attribute is changed from `public` to `protected`. This change restricts the use of `onFailure` to classes derived from `Fixture`.

matlab.unittest.constraints.IsEqualTo always compares table column variables
Behavior change

In release R2019a, the `IsEqualTo` constraint always compares the size and type of column variables.

External Language Interfaces

C++: Use C++ classes from third-party libraries in MATLAB

If you have a library that exports C++ constructs, including classes, functions and enumerations, then you can use this functionality directly in MATLAB. For more information, see C++ Libraries.

If you have a C shared library, then use the `loadlibrary` function as described in C Libraries.

Python: Version 3.7 support

MATLAB now supports CPython 3.7, in addition to existing support for 2.7, 3.5, and 3.6.

For more information, see Install Supported Python Implementation.

Compatibility Considerations

To start the MATLAB engine asynchronously from Python 3.7, use the `(background=True)` keyword argument for `matlab.engine.start_matlab`. To call a MATLAB function asynchronously, use the `background=True` keyword argument for `matlab.engine.MatlabEngine`. Do not use the `async` argument for either function, since it is a keyword in Python 3.7. You also can use the `background` argument for all supported versions of Python.

Python engine: Data type support

The Python engine now supports this functionality:

- Convert MATLAB strings to Python strings
- Pass function handles to Python with the `feval` command
- Pass MATLAB value objects as opaque objects

C++ MEX: Execute MEX function out of process

Run C++ MEX functions in processes that are separate from the MATLAB process. You can run multiple MEX functions in the same process and can create multiple processes to execute MEX functions. For more information, see Out-of-Process Execution of C++ MEX Functions.

MEX functions: Use customer version of Boost library

Although MATLAB builds with Boost library version 1.56.0, as of MATLAB R2018a, you can use any Boost library version in a MEX function.

MATLAB Data Array: Support for row-major memory layout

Create a `matlab::data::Array` with data memory layout specified as column-major (default) or row-major. For more information, see the `memoryLayout` parameter in `createArrayFromBuffer`. To determine the memory layout for an existing `matlab::data::Array`, call `getMemoryLayout`.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2019 with Microsoft Visual Studio 2015 and 2017 for C, C++, and Fortran	Windows
Added	Intel Parallel Studio XE 2019 for Fortran	macOS

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Hardware Support

MATLAB Support Package for Parrot Drones: Control Parrot Mambo FPV drone from MATLAB and acquire sensor data

The MATLAB Support Package for Parrot® Drones is available from release R2019a onwards.

The support package includes functions to pilot a Parrot Mambo FPV drone by sending MATLAB commands to control its direction, speed, and orientation. You can also read the flight navigation data such as speed, height, and orientation using MATLAB commands.

Deploy Sense HAT functions on Raspberry Pi hardware

These Sense HAT functions from the MATLAB Support Package for Raspberry Pi Hardware are enhanced to generate code: `sensehat`, `readHumidity`, `readPressure`, `readTemperature`, `readAngularVelocity`, `readAcceleration`, `readMagneticField`, `readJoystick`, `displayImage`, `writePixel`, and `clearLEDMatrix`. You can now deploy these functions on the hardware.

Functionality being changed or removed

The i2cdev and spidev functions will be removed in a future release
Warns

Use `device` instead of `i2cdev` and `spidev` to connect to I2C or SPI devices on Arduino hardware.

The property Pins of servo object will be removed in a future release
Warns

Use the property `Pin` instead of `Pins` to get the pin number of the Arduino hardware and the Adafruit Motor Shield V2 for Arduino hardware to which the servo motor is connected. For more information, see Connection to servo motor on Arduino and Connection to servo motor on AdafruitMotor Shield V2.

The class arduinoio.LibraryBase will be removed in a future release
Warns

Use the class `matlabshared.addon.LibraryBase` instead of `arduinoio.LibraryBase` for deriving Arduino add-on libraries.

MATLAB support for Adafruit Bluefruit EZ-Link Shield and Programmer will be removed in a future release
Warns

The support for AdafruitBluefruit EZ-Link Shield and Programmer will be removed in a future release

MATLAB support for Arduino hardware boards has been removed
Errors

These Arduino hardware boards are no longer supported:

- Arduino Fio
- Arduino Mini
- Arduino Pro

R2018b

Desktop

Live Editor: Organize live scripts using additional subheading styles

Format text in live scripts using the new **Heading 2** and **Heading 3** text styles. To apply a text style, go to the **Live Editor** tab and in the **Text** section, select any of the options under the **Text Style** drop-down.

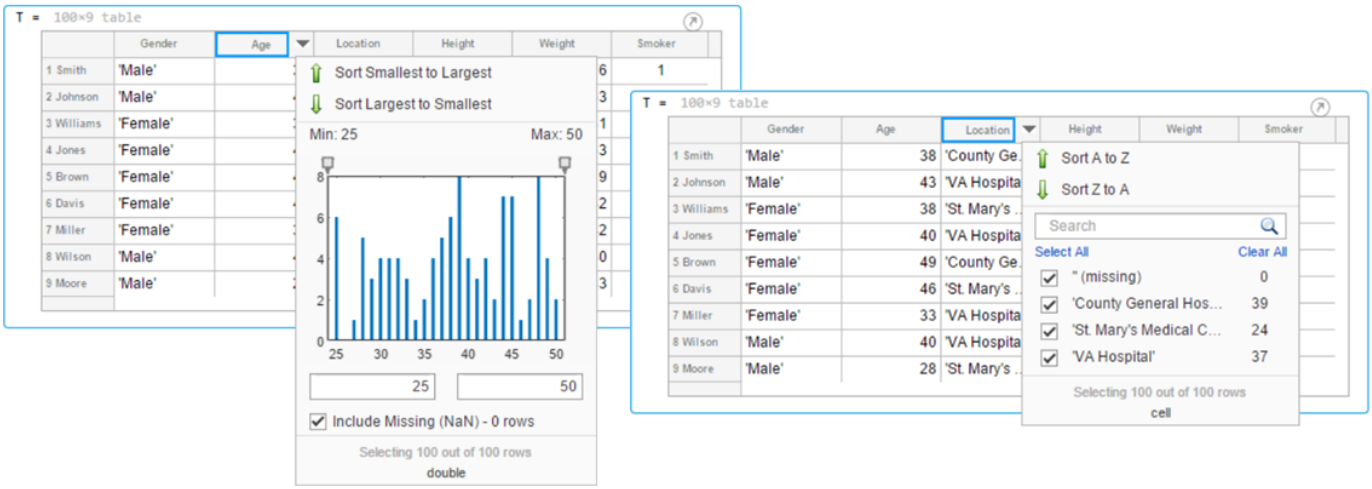
For more information, see Format Files in the Live Editor.

Live Editor: Navigate within a live script using internal hyperlinks

Use internal hyperlinks to navigate to locations within a live script. To insert an internal hyperlink, go to the **Insert** tab and click **Hyperlink**. Enter your display text, select **Internal Hyperlink**, and then click anywhere in the document to select the target.

Live Editor: Filter table output interactively, and then add the generated code to the live script

In the Live Editor, you can filter table data interactively. To filter data in a table, click the down arrow ▾ to the right of a variable name in the table and select from the available filtering options.



To add the generated code to your live script, use the **Update Code** button below the table. Adding the generated code to your live script ensures that the sorting is reproduced the next time you run the live script.

Live Editor: Create new and open existing live scripts faster

New and existing live scripts open faster than in previous releases.

Live Editor: Change case of text or code

In the Live Editor, you can change the case of selected text or code from all uppercase to lowercase, or vice versa. To change the case, select the text, right-click, and select **Change Case**. You also can

press **Ctrl+Shift+A**. If the text contains both uppercase and lowercase text, MATLAB changes the case to all uppercase.

In MATLAB Online, this feature also is available in the Editor.

Comparison Tool: Merge two versions of a live script or function

When comparing live scripts or live functions using the Comparison Tool, you can merge changes from one file to the other. Merging changes can be useful when resolving conflicts between different versions of a file.

To merge two live scripts or functions, go to the **Live Editor** tab and in the **File** section, click **Compare**. A new window opens and displays the two files side by side. Select the  **Merge Mode** button to start the merge.

Use the  button to replace content in the right pane with content from the left pane. The right pane contains the merged result. To save the result, click  **Save Result**.

For more information, see Compare and Merge Live Code.

Add-On Manager: Install and manage multiple versions of a custom toolbox

You can install multiple versions of a custom toolbox in the Add-On Manager. Having multiple versions of a custom toolbox installed is useful if you regularly use multiple versions and switch between them.

To install an additional version of a custom toolbox without overwriting any of the other installed versions, use the `matlab.addons.install` function and specify `'add'` as the installation option. For example, `matlab.addons.install('C:\myAddons\GUI Layout Toolbox 2.1.2.mltbx','add')`.

To select which version of the toolbox is enabled, go to the **Home** tab and select  **Add-Ons** > **Manage Add-Ons**. Click the ⋮ button to the right of the toolbox you want. Then in the **Versions** menu, select from the available versions. Selecting a version enables that version and disables all other installed versions of the toolbox. You also can use the `matlab.addons.enableAddon` function.

Add-On Manager: Save add-ons to new default location

MATLAB now saves add-ons to a new default location. The default location is platform-specific.

- Windows platforms — `C:\Users\`*`username`*`\AppData\Roaming\MathWorks\MATLAB Add-Ons`.
- Linux platforms — `~/MATLAB Add-Ons`.
- Mac platforms — `~/Library/Application Support/MathWorks/MATLAB Add-Ons`.

For more information, see Manage Your Add-Ons.

Documentation: View MATLAB documentation in Spanish

A *subset* of MATLAB documentation in Spanish is available on the web to licensed MATLAB users. For more information, see Translated Documentation.

Language and Programming

string Arrays: Use string arrays in MATLAB, Simulink, and Stateflow

Specify text as string arrays where you previously specified text as character vectors or cell arrays of character vectors. Use string arrays for data, properties, and name-value pair arguments. Specify strings using double quotes, just as you specify character vectors using single quotes.

For more information on string arrays, see Characters and Strings. For guidelines on accepting strings in your own code, see Update Your Code to Accept Strings.

MathWorks encourages the use of string arrays. For backward compatibility, MathWorks products will continue to support the use of character vectors and cell arrays of character vectors.

convertContainedStringsToChars Function: Convert string arrays at any level of cell array or structure

To make your existing code accept string arrays, or cell arrays and structure arrays that contain strings, use the `convertContainedStringsToChars` function on the entire input argument list. For more information on accepting strings in your own code, see Update Your Code to Accept Strings.

Enumerations: Improved performance of set operations with enumerations

When called with enumeration arrays, execution of set operation functions such as `ismember` is faster.

WSDL Web Services Documents: Required Tools Update

As of MATLAB R2018b, the supported versions of the Oracle® Java JDK and the Apache CXF programs that are required to use a WSDL Web service in MATLAB have changed.

For more informations, see Set Up WSDL Tools.

Compatibility Considerations

Download and install the JDK software from the Java SE Downloads Web page https://www.oracle.com/technetwork/java/javase/downloads. Choose the Java SE Development Kit 8.

Download the latest version 3.2 release of the Apache CXF tool from https://cxf.apache.org/download.

Functionality being removed or changed

validateattributes check for 'finite' and 'nonnan' attributes
Behavior change in future release

In the `validateattributes` function, the `'finite'` and `'nonnan'` attributes no longer require that the input passes an `isnumeric` check.

Folders named resources will not be allowed on the MATLAB path
Still runs

In future releases, the `resources` folder will become a reserved folder, and folders with the name `resources` will not be allowed on the MATLAB path. Currently, these folders are allowed on the MATLAB path.

If a folder named `resources` is specified when calling the `addpath`, `userpath`, or `pathdef` functions, MATLAB will return a warning and the folder will not be not added to the path.

Rename all folders on the path named `resources`.

Mathematics

boundaryshape Function: Create a polyshape object from a 2-D triangulation

You now can use the `boundaryshape` function to convert a 2-D `triangulation` object to a `polyshape` object.

polyshape Objects: Specify when to keep collinear points when creating a polyshape

When creating a `polyshape` object, collinear points are removed by default. These functions now offer the option to keep collinear points as vertices of the returned `polyshape` using the name-value pair `'KeepCollinearPoints'`.

`polyshape`	`subtract`
`addboundary`	`union`
`intersect`	`xor`
`simplify`	

RandStream Objects: Generate random numbers using Threefry and Philox algorithms

When creating a random stream with `RandStream`, you now can use the Threefry and Philox random number generation algorithms.

GraphPlot Object: Customize node and edge labels with font properties

`GraphPlot` objects have several new properties to enable customization of node and edge labels in plots of directed or undirected graphs.

Property	Description

`NodeLabelColor`
 	
`EdgeLabelColor`
	Background color of label

`NodeFontSize`
 	
`EdgeFontSize`
	Font size for label

`NodeFontName`
 	
`EdgeFontName`
	Font for label

`NodeFontAngle`
 	
`EdgeFontAngle`
	Normal or italic text

Property	Description

`NodeFontWeight`
 	
`EdgeFontWeight`
	Normal or bold text

`Interpreter`	Interpretation of text characters in labels (none, tex, or latex)
`ArrowPosition`	Position of arrow on directed edges

Compatibility Considerations

The new `GraphPlot` property `Interpreter` has a default value of `'tex'`. In previous releases, graph node and edge labels displayed text as the literal characters instead of interpreting the text using TeX markup. If you do not want node and edge labels to use TeX markup, then set the `Interpreter` property to `'none'`.

sinpi and cospi Functions: Compute the sine and cosine of multiples of π

The `sinpi` and `cospi` functions compute the values of sin(*πx*) and cos(*πx*). The answers provided by these functions are more accurate than answers provided by `sin(pi*x)` or `cos(pi*x)` because they do not compute `pi*x` explicitly. This convention compensates for roundoff error in the floating-point value of `pi`.

Graphics

Axes Interactions: Explore data with panning, zooming, data tips, and 3-D rotation enabled by default

Interactively explore your data using axes interactions that are enabled by default. For example, you can use the scroll-wheel to zoom into your data or hover over a data point to see a data tip. Also, you can click and drag the axes to pan the axes (2-D view) or rotate the axes (3-D view). For more information, see Interactively Explore Plotted Data.

Compatibility Considerations

In previous releases, none of the interactions were enabled by default. To control if the axes interactions are enabled by default, use the `disableDefaultInteractivity` and `enableDefaultInteractivity` functions.

Axes Toolbar: Access and customize a data exploration toolbar for each Axes object

Axes have a toolbar that appears above the top-right corner for quick access to the data exploration tools. The buttons available in the toolbar depend on the contents of the axes. The toolbar typically includes buttons to brush data, add data tips, rotate the axes (3-D axes only), pan or zoom the data, and restore the view.

You can customize the buttons available in the toolbar using the `axtoolbar` and `axtoolbarbtn` functions.

Compatibility Considerations

In previous releases, the buttons that now appear in the axes toolbar appeared in the figure toolbar instead. You can turn off the axes toolbar by setting the `Visible` property of the `AxesToolbar` object to `'off'`.

```
ax = gca;
ax.Toolbar.Visible = 'off';
```

You can restore the figure toolbar buttons using the `addToolbarExplorationButtons` command.

Geographic Plots: Create line, scatter, and point density plots on interactive maps and control properties of a geographic axes

Create line, scatter, and point density plots on interactive maps and control properties of a geographic axes. Use the `geoplot`, `geoscatter`, and `geodensityplot` functions to create these plots. The `geolimits` function now works with any of these geographic plots, in addition to geographic bubble charts. To change the basemap used by any of these geographic plots or charts, use the new `geobasemap` function.

stackedplot Function: Plot variables of a table or timetable for comparison using a common x-axis

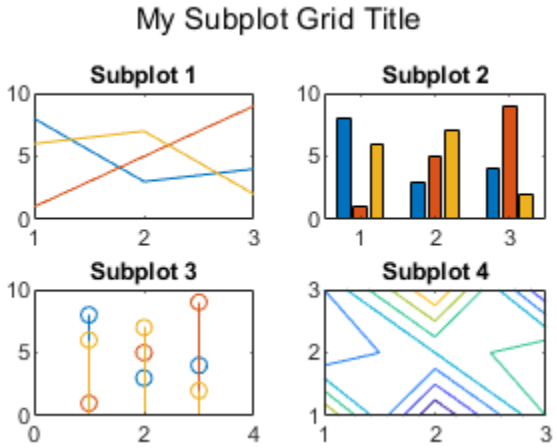
Plot the variables of a table or timetable. To ease visual comparison, the `stackedplot` function provides a common *x*-axis and separate *y*-axes for the variables.

scatterhistogram Function: Visualize grouped data as a scatter plot with marginal histograms

To create a scatter plot with marginal histograms, use the `scatterhistogram` function. To group the data, you can use either the `'GroupVariable'` name-value pair argument with tabular data or the `'GroupData'` name-value pair argument with arrays.

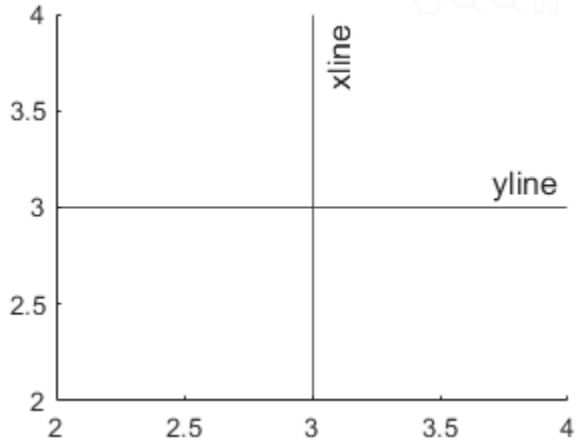
sgtitle Function: Create a title for a grid of subplots

You can add an overall title to a grid of subplots, in addition to adding a title to each individual subplot. To add an overall title, use the `sgtitle` function.



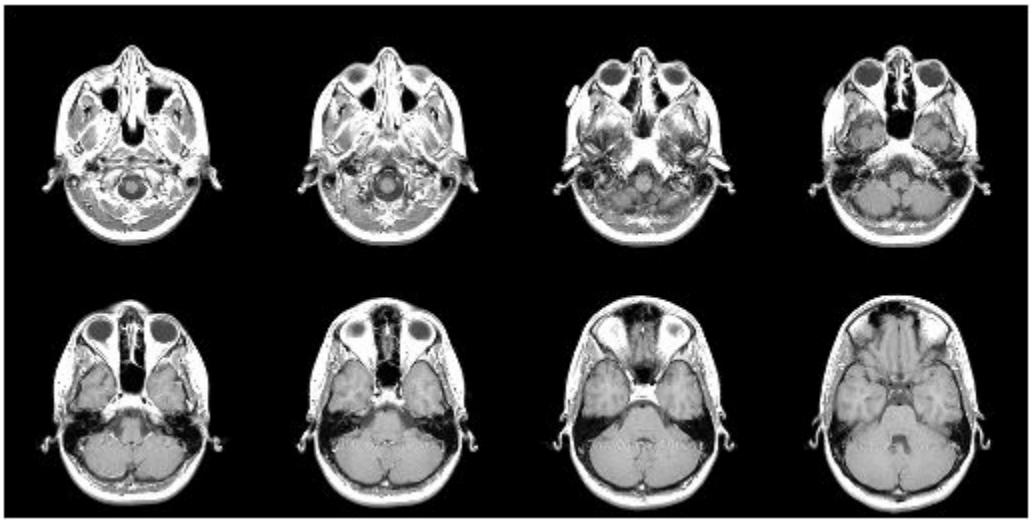
xline and yline Functions: Add vertical or horizontal lines to a plot

To add vertical or horizontal lines to a plot, use the `xline` or `yline` functions, respectively. For example, `xline(3)` plots a vertical line at x = 3.



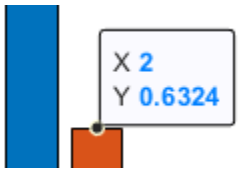
imtile Function: Combine multiple image frames into one rectangular tiled image

To combine multiple image frames into one rectangular tiled image, use the `imtile` function.



Data Tips: Use TeX or LaTeX markup in data tips with improved visual appearance

Data tips have an improved visual appearance with new text colors.



Also, data tips now display text characters using TeX markup by default. Control the interpretation of the text characters using the `Interpreter` property of the data cursor mode object. Set the property

value to `'tex'` for TeX markup (default), `'latex'` for LaTeX markup, or `'none'` for literal characters.

```
d = datacursormode;
d.Interpreter = 'latex';
```

Compatibility Considerations

In previous releases, data tips displayed text as the literal characters instead of interpreting the text using TeX markup. If you do not want data tips to use TeX markup, then set the `Interpreter` property to `'none'`.

Functionality being removed or changed

legend function interprets argument as property name when property exists

Behavior change

Starting in R2018b, if you pass an argument to the `legend` function that matches the name of a legend property, the function interprets the argument as the name of a name-value pair. In previous releases, the `legend` function recognized name-value pairs only when the first argument was a cell array.

As a result of this change, in most cases, it is unnecessary to specify the first argument as a cell array when using name-value pairs. However, if you want a label in your legend that matches the name of a legend property, such as `Position` or `NumColumns`, then you *must* specify all the labels in a cell array. Otherwise, the `legend` function interprets the argument as a name-value pair instead of a label.

Description	Recommended Code
If you want a label in your legend that matches the name of a legend property, such as `'NumColumns'`, then specify all the labels in a cell array. If you specify `'NumColumns'` outside of a cell array, the `legend` function interprets it as a name-value pair.	`legend({'Label1','NumColumns','Label3','Label4'},'Num`
If none of your labels match the name of a legend property, then you do not need to use a cell array around the labels.	`legend('Label1','Label2','Label2')`

alpha and shading set both FaceColor and FaceAlpha properties

Behavior change

When updating surface and patch objects, the `alpha` and `shading` functions sometimes set both the `FaceColor` and `FaceAlpha` properties. These functions set both properties in cases where setting just one property results in a rendering issue. No updates to your code are required.

In previous releases, the `alpha` function set only the `FaceAlpha` property. Similarly, the `shading` function set only the `FaceColor` property.

Data Import and Export

Import Tool: Generate improved code when importing from spreadsheets

The **Import Tool** now offers improved code generation functionality for importing spreadsheets across platforms. For example, you can import datetimes on Mac and Linux and generate code that is easy to read. For more information, see Read Spreadsheet Data Using Import Tool.

Compatibility Considerations

For more information, see “Import Tool handling of spreadsheet dates and times and fields that are empty, unimportable, or error causing” on page 4-14.

Web-Based Data: Read from web-based data sources like Amazon Web Services and Azure Blob Storage using readtable, detectImportOptions, spreadsheetDatastore, imread, and imfinfo

You can access tabular data and images from files stored in remote locations (Amazon S3, Windows Azure® Blob Service, and HDFS™) using these functions:

- `readtable`
- `detectImportOptions`
- `spreadsheetDatastore`
- `imread`
- `imfinfo`

When reading data from remote locations, you must specify the full path using a uniform resource locator (URL). For example, read a `csv` file from Amazon S3 cloud:

```
T = readtable('s3://bucketname/path_to_file/my_text_file.csv');
```

For more information on setting up MATLAB to access your online storage service, see Work with Remote Data.

write Function: Write tall arrays in a variety of formats to local or remote locations

The functionality of `write` is expanded to support additional formats and storage locations:

- Output formats include `.txt`, `.csv`, `.xls`, and more.
- Name-value pairs to control format-specific options, such as `WriteVariableNames` and `Delimiter`.
- Extended support for writing `.seq` and `.mat` formats to all supported file systems.
- Write data to remote locations in Amazon S3 or Windows Azure Blob Storage (WASBS).

stlread and stlwrite Functions: Read from and write to STL (Stereolithography) files for triangulations

The `stlread` function enables you to read triangulation information from an STL file to create a `triangulation` object. You also can write a `triangulation` object or a 2-D `delaunayTriangulation` object to a binary STL file with `stlwrite`.

TabularTextDatastore Object: Import data containing dates and times from non-English locales

The `TabularTextDatastore` object now supports import of date and time data from non-English locales. For example, to create a datastore for reading files that contain German date and time data, set the `DatetimeLocale` parameter to `'de_DE'`.

```
ds = tabularTextDatastore('myDataFile.csv','DatetimeLocale','de_DE')
```

readtable and writetable Functions: Read or write spreadsheet files without initiating Microsoft Excel for Windows on Windows platforms

On Windows platforms, you can choose not to open an instance of Microsoft Excel when reading or writing data from spreadsheet files. Set the `'UseExcel'` parameter to one of these values:

- `true` — Open an instance of Microsoft Excel to read (or write) the file. This setting is the default for Windows systems with Excel installed.
- `false` — Do not open an instance of Microsoft Excel to read (or write) the file. Using this setting might cause the data to be written differently for files with live updates like formula evaluation or plugins.

For more information, see `readtable` and `writetable`.

readtable Function: Manage the import of empty fields using import options

You can manage empty fields in tabular data by using `readtable` along with import options. Use the `EmptyFieldRule` of import options object to specify how `readtable` handles empty fields. For more information, see `setvaropts`.

Scientific File Format Libraries: CFITSIO Library upgraded to version 3.420

The CFITSIO library is upgraded to version `3.420`.

Functionality being removed or changed

Import Tool handling of spreadsheet dates and times and fields that are empty, unimportable, or error causing

Behavior change

Starting in R2018b, the **Import Tool** app has improved functionality for importing data from spreadsheet files. The changes to imported data are minimal and are limited to uncommon cases:

- Empty or unimportable cells in spreadsheets:
  - The value you use to replace empty cells and unimportable cells must be the same. Previously, **Import Tool** allowed for different values for empty cells and unimportable cells.
  - Previously, **Import Tool** allowed you to specify a target string for cells that are unimportable. This feature is no longer supported.
- Date and time data in spreadsheets:
  - Import date and time data as MATLAB `datetime` arrays on all platforms.
  - Import numbers as date and times on all platforms.
  - Import Excel dates as numbers on all platforms.
- When running the **Import Tool** app in MATLAB on a Windows machine, cells in Excel spreadsheets with error conditions are no longer displayed.

Basic parameter of the readtable function (Not Recommended)
Still runs

The `Basic` parameter of the `readtable` function is not recommended. Use the parameter name `UseExcel` instead. There are no plans to remove the `Basic` parameter at this time.

UseExcel parameter of the readtable and writetable functions
Behavior change in future release

In future releases, the default value of the `UseExcel` parameter will be changed to `false`. The current default setting for `UseExcel` on Windows systems with Excel installed is `true`.

This table shows the typical usage of `readtable` and how to update your code to preserve the current behavior in future releases.

Code (R2018b and earlier)	Use this instead (Future releases)	Behavior
`T = readtable(filename)`	`T = readtable(filename,'UseExcel',true)`	Start an instance of Microsoft Excel when reading the file.

For more information, see `readtable` and `writetable`.

Output from the audioread function for A-law or mu-law wave files
Behavior change

When reading in native mode, the `audioread` function returns data from A-law or mu-law wave files as `int16`.

Previously, `audioread` returned data from A-law or mu-law wave files as `int8`.

Transparency output from the imread function
Behavior change

The `imread` function returns transparency information for indexed PNG images. For example, reading a png file returns a nonempty transparency array `alpha`:

```
[img,map,alpha] = imread('myIndexedImage.png');
whos alpha
```

```
Name        Size             Bytes  Class     Attributes

  alpha      50x120            48000  double
```

Previously, the `imread` function did not return transparency information for indexed PNG images. For example, previously reading a png file returned an empty transparency output `alpha`:

```
[img,map,alpha] = imread('myIndexedImage.png');
whos alpha
```

```
Name       Size            Bytes  Class     Attributes

  alpha      0x0                0  double
```

Data Analysis

Vector Dimension Argument: Operate on multiple dimensions at a time for selected reduction functions

These functions now accept a vector dimension argument to specify multiple operating dimensions at a time, as well as the option `'all'` to specify all dimensions of an array.

`all`	`min`
`any`	`mode`
`bounds`	`prod`
`max`	`std`
`mean`	`sum`
`median`	`var`

For example, `sum(A,'all')` sums all the elements in a matrix `A`, and is equivalent to `sum(A,[1 2])`.

grouptransform Function: Transform table or timetable data by groups

You can use the `grouptransform` function to perform group computations, such as normalization or filling missing data on table and timetable variables. For example, `g = grouptransform(T,'School','norm')` normalizes the data in a table `T` by school using the vector 2-norm.

groupsummary Function: Perform group summary computations on matrices

You now can group by matrix rows to perform summary computations using the `groupsummary` function.

tall Arrays: Write custom algorithms to operate on tall arrays

The functions `matlab.tall.transform` and `matlab.tall.reduce` enable you to write custom algorithms to execute on tall arrays. These functions enable you to implement a range of parallelizable algorithms. `matlab.tall.transform` applies a single function to each block of a tall array, while `matlab.tall.reduce` is similar to MapReduce, where two functions are applied to a tall array, with the output of the first function being fed as input to the second function.

`matlab.tall.transform` and `matlab.tall.reduce` provide the flexibility to implement functions that otherwise do not currently support tall arrays. For more information, see Develop Custom Tall Array Algorithms.

tall Arrays: Operate on tall arrays with more functions, including conv2, wordcloud, and groupsummary

The functions listed in this table add support for tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, see the **Extended Capabilities** section at the bottom of the reference pages.

`conv2`	`islocalmin`
`convn`	`rmoutliers`
`corrcoef`	`vecnorm`
`groupsummary`	`wordcloud`
`islocalmax`	

In addition, some functions have expanded support for tall arrays. This expanded support removes some limitations of using these functions with tall arrays.

Functions	Added Support
`join`	Joining two tall inputs (tall tables and/or tall timetables) is now supported. *Previously, one of the inputs was required to be an in-memory table or timetable.*

`sort`
 	
`sortrows`
 	
`topkrows`
 	
`unique`
	These functions now support multiple output arguments. The extra outputs return indices that are relevant to the operation or that describe the location of interesting elements.
	
	Previously, these functions did not support multiple outputs for tall arrays.

`mean`	Calculating the mean of datetime values is now supported. *Previously, this function did not support tall arrays with an underlying data type of datetime.*

rmoutliers Function: Remove outliers in an array, table, or timetable

The `rmoutliers` function detects and removes outlier data in an array, table, or timetable.

islocalmin and islocalmax Functions: Specify a range of data for prominence computation

The `'ProminenceWindow'` name-value pair for the `islocalmin` and `islocalmax` functions enables you to specify a local neighborhood about each element in the input data when computing the corresponding prominence.

Table and Timetable Metadata: Store custom metadata for each variable

You can store custom metadata for variables of a table or timetable using its `Properties.CustomProperties` object. For more information, see the Custom Metadata sections of `table` and `timetable`.

timetable Data Type: Save memory when storing row times with regular time steps

If you create a timetable with regular time steps, then it stores the row times using less memory than in previous releases. You can create a regular timetable using the `array2timetable`, `retime`, `synchronize`, `table2timetable`, or `timetable` functions.

A timetable stores the start time, time step, and sample rate as properties. For more information, see the Row Times Metadata section of `timetable`.

timerange Function: Specify unit of time to define time range

When you specify the beginning and end of a time range using datetime values, you also can specify the date or time component used to define the endpoints. For example, `S = timerange('2018-9-30','quarters')` creates a time range spanning all dates in the third quarter of 2018, since September 30 is in the third quarter. For more information, see `timerange`.

convertvars Function: Convert table or timetable variables to specified data type

You can specify variables of a table or timetable, and convert them to a different data type, using the `convertvars` function.

table, timetable, and addvars Functions: Use single quotes for input names, not double-quoted strings

When creating or updating a table or timetable using the `table`, `timetable`, or `addvars` functions, use single quotes for input names (such as `'Size'`, `'VariableNames'`, `'After'`, or `'TimeStep'`) to avoid confusion with variable inputs. Variables and input values can use double-quoted strings. For example:

```
T = table("some text",'VariableNames',["X"]);
```

Functionality Being Removed or Changed

'SamplingRate' is not recommended
Still runs

The `'SamplingRate'` name-value pair argument is not recommended. Use `'SampleRate'` instead. The corresponding timetable property is also named `SampleRate`.

For backward compatibility, you still can specify `'SamplingRate'` as the name of the name-value pair. However, the value is assigned to the `SampleRate` property.

This change in behavior affects the timetable functions shown in the table.

Function
`array2timetable`
`retime`
`synchronize`
`table2timetable`
`timetable`

Default random number generator change for tallrng
Behavior change in future release

Starting in R2019a, the default random number generator for `tallrng` will change to `threefry`. This generator offers performance enhancements for parallel calculations over the current default. In releases up to R2018b, the default random number generator for `tallrng` is `combRecursive`.

With a different default generator, MATLAB will generate different sequences of random numbers by default in the context of tall arrays. However, the statistics of these calculations will remain unaffected. Therefore, you should update any code that relies on the specific random numbers being generated. However, most calculations on the random numbers should be unaffected.

To set the generator to the settings used by default in R2018b and earlier releases, use the command:

```
tallrng(0,'combRecursive')
```

App Building

App Designer: Add and configure date selection components on the App Designer canvas

Drag and drop date picker components from the **Component Library** onto the canvas.

App Designer: Unified property inspector in Design View and Code View

The **Component Properties** pane in Design View now provides the full list of properties and the same interactive controls as Code View does.

App Designer: Expand and collapse sections of code in Code View

App Designer provides code folding capabilities in Code View. You can expand sections of code that you are working on and collapse other sections to navigate your code more easily.

App Designer: Export apps as code files

Export apps you create in App Designer as (.m) program files. Select **Save > Export to .m File** in the App Designer toolstrip.

Exporting an app as a program file enables you to change it outside of App Designer. However, there is no option for importing your changes back into App Designer.

App Designer: Locate errors and warnings in your code with the Code Analyzer message bar

App Designer now provides the same Code Analyzer messaging system as the MATLAB Editor does.

App Designer: Program apps faster using improved code suggestions and completions

As you code your app, App Designer displays the same contextual hints for arguments, property values, and syntaxes as the Live Editor does.

App Designer: Control App Designer Code View settings using MATLAB preferences

Control the following settings for App Designer Code View by setting MATLAB preferences:

- Highlight current line
- Show line number
- MATLAB syntax highlighting colors

When you set any of these preferences, the change applies to the MATLAB Editor and App Designer.

uigridlayout Function: Configure app layouts using a grid layout manager

Use the `uigridlayout` function to create grid layout managers in your apps. When you use a grid layout manager, you position UI components along the rows and columns of an invisible grid that spans the entire app window or a container in the window. Using rows and columns to position components is easier to manage than setting pixel values in `Position` vectors. For resizable apps, grid layouts provide more flexibility than the automatic resize behavior in App Designer, and they are easier to code than `SizeChangedFcn` callbacks. For more information, see Using Grid Layout Managers.

Grid layout managers are only available in apps created with the `uifigure` function.

Scrolling Containers: Enable scrolling for figure, panel, tab, and button group containers

Enable scrolling within a figure or any child container by setting the `Scrollable` property to `'on'`. To scroll to a location within the container programmatically, call the `scroll` function.

Scrolling works only in App Designer apps, in figures created with the `uifigure` function, or in child containers within those figures.

Figure Interactions: Create apps with custom mouse and keyboard interactions using figures created with the uifigure function

Create apps that respond to custom mouse and keyboard interactions. The following properties are available in App Designer and in figures created with the `uifigure` function.

Category	Properties
Figure Data	`SelectionType`, `CurrentPoint`, `CurrentCharacter`
Mouse Callbacks	`ButtonDownFcn`, `WindowButtonDownFcn`, `WindowButtonUpFcn`, `WindowButtonMotionFcn`, `WindowScrollWheelFcn`
Keyboard Callbacks	`KeyPressFcn`, `KeyReleaseFcn`, `WindowKeyPressFcn`, `WindowKeyReleaseFcn`

Figures created with the `figure` function continue to support these properties as they have in the past.

Graphics Support: Integrate plots into an app using the axes, polaraxes, and geoaxes functions

Display more types of plots in App Designer apps and in figures created with the `uifigure` function. The expanded list of plots includes subplots, polar plots, and geographic plots. For more information, see Displaying Graphics in App Designer.

Tooltips: Create custom tooltips for UI components in apps

Set the `Tooltip` property on a UI component to display a tooltip when the user hovers the mouse over the component at run time. Tooltips display even when the components are disabled.

The `Tooltip` property is available for UI components in App Designer apps and in figures created with the `uifigure` function.

If you are creating an app using GUIDE or the `figure` function, use the `Tooltip` property instead of the `TooltipString` property on `UIControl`, `Table`, `Tab`, `PushTool`, and `ToggleTool` UI components. For details, see “TooltipString property is not recommended” on page 4-23.

Deployed Web Apps: Access files in deployed web apps using the uigetfile and uiputfile functions

Call `uigetfile` in a web app to enable users to select files on their local systems. Users can select files by dropping them onto the app or by selecting them in a file browser.

Call `uiputfile` to enable users to specify file names for saving files.

For information about other new features of deployed web apps, see Release Notes (MATLAB Compiler).

Running Apps in Browsers: Use most modern browsers to run apps in MATLAB Online or as deployed web apps

Run apps in MATLAB Online, and run deployed web apps using Safari, Firefox, and Microsoft Edge in addition to Google Chrome and Internet Explorer. Use the current versions of these browsers for best results. For the best overall experience, use Google Chrome.

For information about MATLAB Online system requirements, see https://www.mathworks.com/products/matlab-online/system-requirements.html.

For information about other new features of deployed web apps, see Release Notes (MATLAB Compiler).

uisetcolor Function: Select custom colors interactively

Select custom colors using an improved interactive color picker in the `uisetcolor` dialog box. Make your selection by clicking within a color gradient. You can select colors as HSV values in addition to the RGB and hexadecimal options that have been available in previous releases.

Functionality Being Removed or Changed

TooltipString property is not recommended

Still runs

The `TooltipString` property for the `UIControl`, `Table`, `Tab`, `PushTool`, and `ToggleTool` UI component objects is not recommended. Use the new `Tooltip` property to display a tooltip instead.

The `TooltipString` property is no longer listed when you call the `get`, `set`, or `properties` functions to list the properties of the object. There is no plan at this time to remove support for getting or setting the value of the `TooltipString` property. However, partially specifying the `TooltipString` property name might produce errors. To prevent the error, specify the full name of the `TooltipString` property, or use the `Tooltip` property. Both property names correspond to the same value.

Performance

Startup: Increased speed of MATLAB startup

MATLAB starts faster because of continued infrastructure improvements and optimizations.

Execution Engine: Index into large arrays with improved performance when using the colon operator

Now colon indexing into large, numeric arrays is faster.

Execution Engine: Faster calls to built-in functions

Calls to built-in functions are faster due to reduced overhead.

Live Editor: Create new and open existing live scripts faster

New and existing live scripts open faster than in previous releases.

Enumerations: Improved set function performance with enumerations

When called with enumeration arrays, execution of set operation functions such as `ismember` is faster.

Building Apps: Faster canvas interactions in App Designer

Many common tasks in the App Designer canvas, such as copying, pasting, and deleting components are 90% faster.

Running Apps: Faster startup time for apps

Startup time is 10% to 30% faster for apps created in App Designer and apps created programmatically using the `uifigure` function. The time savings become more noticeable as the number of components in your app increases.

sort Function: Sort matrices and arrays faster

The `sort` function is now faster when sorting numeric matrices or multidimensional arrays.

Hardware Support

MATLAB Online: Communicate with Raspberry Pi hardware board from MATLAB Online

You now can connect to and control Raspberry Pi hardware boards remotely from MATLAB Online. Raspberry Pi 2 Model B and Raspberry Pi 3 Model B are supported. Install the MATLAB package onto Raspberry Pi, use `raspilist` in MATLAB Online to discover available boards, and use `raspi` to create a connection. The functions in the MATLAB Support Package for Raspberry Pi Hardware (except `openShell` and `putFile`) are available in MATLAB Online. For more information on how to communicate with your Raspberry Pi in MATLAB Online, see Connect to Raspberry Pi Hardware Board in MATLAB Online (MATLAB Support Package for Raspberry Pi Hardware).

Deploy a MATLAB function on Raspberry Pi hardware

From R2018b, MATLAB Support Package for Raspberry Pi Hardware enables you to deploy your MATLAB function as a standalone executable on the Raspberry Pi hardware. For deploying the function, use the `targetHardware` command to create a Raspberry Pi configuration object, and then use the `deploy` command to deploy the function on the Raspberry Pi hardware.

To use this feature, you must install MATLAB Coder in your computer.

To support deployment, the Raspberry Pi functions are enhanced to generate code. Some functions are listed here. For more details, see MATLAB Support Package for Raspberry Pi Hardware.

Peripheral	Function
Raspi	`raspi`
LEDs	`writeLED`
GPIO Pins	`configurePin`, `readDigitalPin`, and `writeDigitalPin`
I2C Interface	`i2cdev`, `read`, `write`, `readRegister`, and `writeRegister`
SPI Interface	`spidev` and `writeRead`
Serial Port	`serialdev`, `read`, and `write`
Servo	`servo` and `writePosition`
Linux	`system`
Camera Board	`cameraboard`, `snapshot`, `record`, and `stop`
Web Camera	`webcam` and `snapshot`
Pulse Width Modulation	`writePWMFrequency`, `writePWMDutyCycle`, and `writePWMVoltage`

iOS and Android Sensors: Acquire sensor data when your device does not have network access

You can acquire sensor data locally on your Android™ or Apple iOS device, with or without a network connection. This method is an alternative method of collecting the sensor data instead of streaming it

from the device to your computer running MATLAB. It is especially useful if you want to collect sensor data while your device does not have a network connection.

To use this method of acquiring sensor data, you log sensor data locally on your mobile device using MATLAB Mobile, and then upload the files to MATLAB Drive once you are connected. To use MATLAB Drive, you must log into your MathWorks account. Alternatively, you can transfer the log files manually using a USB cable. Once you have the sensor data on your computer running MATLAB, you use the MATLAB Support Package for Android Sensors or the MATLAB Support Package for Apple iOS Sensors to view and analyze the data.

For more information, see the help screens in MATLAB Mobile or the MATLAB Support Package for Android Sensors or MATLAB Support Package for Apple iOS Sensors documentation.

iOS and Android Sensors: Upload sensor logs from the device to MATLAB Drive

You can acquire sensor data locally on your Android or Apple iOS device, with or without a network connection. You can then upload the files to MATLAB Drive when you are connected. To use MATLAB Drive, you must log into your MathWorks account.

You can have the log files automatically upload to MATLAB Drive when you have a network connection, or choose to upload them from the **Sensor Logs** screen in MATLAB Mobile any time. Go to **Settings > Configure > MATLAB Drive Upload > Auto Upload** to select your preference.

For more information, see the help screens in MATLAB Mobile or the MATLAB Support Package for Android Sensors or MATLAB Support Package for Apple iOS Sensors documentation.

Advanced Software Development

Tab Completion: Validate function signature file with validateFunctionSignaturesJSON function

Use the `validateFunctionSignaturesJSON` function to validate the JSON-formatted file that contains information about your function signatures. For more information, see `validateFunctionSignaturesJSON`.

MATLAB uses the information in `functionSignatures.json` to improve interactive features, such as tab completion and function hints. For information on creating a `functionSignatures.json` file, see Customize Code Suggestions and Completions.

Tab Completion: JSON parser for functionSignatures.json upgrade

The JSON file parser that MATLAB uses to read `functionSignatures.json` files is upgraded for R2018b.

Compatibility Considerations

The updated parser has stricter validation for JSON files. Before R2018b, a `functionSignatures.json` file could have syntax errors that were undetected by the JSON parser. For these newly detected errors, MATLAB displays an error message in the Command Window when it reads the file.

Correct any syntax errors in the JSON file. Best practice is to validate `functionSignatures.json` files with the `validateFunctionSignaturesJSON` function.

Java SE 8: MATLAB support, providing improved security and access to new Java features

Java interface supports JRE version Java 1.8.0_152. For more information, see MATLAB Supported Interfaces to Other Languages.

Python Interface: Pass multidimensional numeric or logical arrays between MATLAB and Python

MATLAB auto converts numeric and logical array data input to a Python function to a Python `memoryview` object. For more information, see Passing Matrices and Multidimensional Arrays.

C++ MEX API: Call MATLAB asynchronously from within a MEX file using the C++ API

Use the asynchronous C++ MEX API to call MATLAB functions from user application threads in MEX functions. Calls to MATLAB from user application threads are queued and executed in sequence with other MATLAB commands. For more information, see Call MATLAB from Separate Threads in MEX Function.

Unit Testing Framework: Run tests in parallel with more plugins and more intelligent scheduling

R2018b includes refinements to the testing framework parallel scheduling algorithm. These enhancements improve the overall performance of the `'UseParallel'` option in `runtests` and the `TestRunner.runInParallel` method.

Also, the following plugins now can run tests in parallel:

- `CodeCoveragePlugin` with Cobertura format
- `TestReportPlugin`
- `XMLPlugin.producingJUnitFormat`

Unit Testing Framework: Use external parameters in parameterized test

You can inject variable inputs into your parameterized test. For example, you can specify that a test uses input data from a file instead of the data hard-coded within a test. To define external parameters, use the `fromData` method of the `matlab.unittest.parameters.Parameter` class. Then, specify that your parameterized test use the external parameters using the `'ExternalParameters'` option to `TestSuite` creation methods. `TestSuite` creation methods include `fromClass`, `fromFile`, `fromFolder`, `fromMethod`, `fromName`, and `fromPackage`.

For more information, see Use External Parameters in Parameterized Test.

Unit Testing Framework: Sort test suite based on shared fixtures

To reduce shared fixture setup and teardown operations, sort test suite elements so that elements that require the same shared fixture setup are adjacent. To sort an existing test suite, use the `sortByFixtures` method of `matlab.unittest.TestSuite`. The `testsuite` function automatically creates a test suite that is sorted based on shared fixtures. However, if you concatenate test suites after creating them, call the `sortByFixtures` method to reorder the suite. For more information, see `matlab.unittest.TestSuite.sortByFixtures`.

Unit Testing Framework: Explicitly control output display detail and logged diagnostic level

From the Run Tests section in the Editor, you can control the amount of detail displayed for a test run. For example, to display the most information, select **Verbose** from the Output Detail test option under the **Run Tests** icon. To suppress output, select **None**.

You can control which logged diagnostics are displayed by selecting a value from the Logging Level test option under the **Run Tests** icon. Logged diagnostics are diagnostics that you supply in your test code with a call to the `TestCase.log` method. MATLAB reports logged diagnostics at the specified logging level and lower. For example, to exclude diagnostics logged at detailed or verbose levels, select **Concise**.

You can also control the output detail and logged diagnostic level programmatically using the `'OutputDetail'` and `'LoggingLevel'` name-value pairs in these features:

- `runtests` function
- `TestRunner.withTextOutput` method
- `DiagnosticsOutputPlugin` class
- `DiagnosticsRecordingPlugin` class
- `TAPPlugin` class
- `XMLPlugin` class (`'OutputDetail'` only)
- `TestReportPlugin` class (`'LoggingLevel'` only)

The `matlab.unittest.Verbosity` enumeration now contains the `Verbosity.None` member. Use this verbosity level to indicate a detail level that includes no information. This enumeration member is accepted anywhere that accepts a `Verbosity` value, except for the `matlab.unittest.TestCase.log` and `matlab.unittest.fixtures.Fixture.log` methods. These methods direct the framework to log diagnostics, not to display them.

Compatibility Considerations

Before R2018b, the `'Verbosity'` name-value pair controlled both the output detail and the logged diagnostic level, and the `'ExcludingLoggedDiagnostics'` name-value pair determined whether plugins recorded logged diagnostics.

These name-value pairs are supported, but are not recommended. Use `'LoggingLevel'` and `'OutputDetail'` instead. Replace instances of `'ExcludingLoggedDiagnostics'` and `'Verbosity'` in the following plugins:

- `runtests` function (`'Verbosity'` only)
- `matlab.unittest.TestRunner.withTextOutput` method (`'Verbosity'` only)
- `matlab.unittest.plugins.DiagnosticsRecordingPlugin` class
- `matlab.unittest.plugins.TAPPlugin` class
- `matlab.unittest.plugins.TestReportPlugin` class

Unit Testing Framework: Configure detail level of output diagnostics

To configure the amount of detail included in an output stream for diagnostics from passing, failing, and logging events, add the `DiagnosticsOutputPlugin` to a `TestRunner` instance. For example, you can specify that Command Window output includes passing diagnostics at a verbose detail level or you can suppress the display of diagnostics.

For more information, see the `matlab.unittest.plugins.DiagnosticsOutputPlugin` class.

Unit Testing Framework: Compare values faster when using constraints

The `matlab.unittest.constraints.IsEqualTo` class has improved performance when comparing equal values.

Compatibility Considerations

Before R2018b, the `IsEqualTo` constraint called `isequal` or `isequaln` to determine if the actual and expected objects were equal. With release R2018b, the constraint can, sometimes, determine

that the actual and expected objects are equal without calling these functions. In these cases, if the objects being compared overload the `isequal` or `isequaln` functions, then whatever specialized behavior these methods define is not used in the comparison.

App Testing Framework: Programmatically choose tree node

The `choose` method now supports programmatic selection of tree nodes in your app tests. For more information, see `matlab.uitest.TestCase.choose`.

Performance Testing Framework: Measure execution time of fast code more accurately with the TestCase.keepMeasuring method

Performance tests that execute too quickly for MATLAB to time accurately are filtered with an assumption failure. With the `keepMeasuring` method, the testing framework can measure significantly faster code by automatically determining the number of times to iterate through code and measuring the average performance. Use the `keepMeasuring` method within the condition of a `while` loop. For more information, see `matlab.perftest.TestCase.keepMeasuring` and Measure Fast Executing Test Code.

Mocking Framework: Invoke function upon mocked method call

You can specify that, each time you call a mocked method, it calls another function. For example, specify that each time you call a mocked `roll` method, it calls the `randi` function.

To specify that a mock method uses a function handle to invoke another function, define behavior with the `Invoke` class of the `matlab.mock.actions` package. This action differs from the `AssignOutputs` action, which returns values that are defined when you create the `AssignOutputs` instance.

For more information, see `matlab.mock.actions.Invoke`.

Mocking Framework: Verify interactions on mock occurred in order

You can create a constraint that is satisfied if interactions with a mock occurred in a specific order. Use the `'RespectingOrder'` option with `matlab.mock.constraints.Occurred` to verify that mock methods were called and properties were accessed and set in a particular order.

For more information, see `matlab.mock.constraints.Occurred`.

Mocking Framework: Clear history of recorded mock object interactions

Use the `clearMockHistory` method of `matlab.mock.TestCase` to clear the history of recorded mock object interactions. For more information, see `matlab.mock.TestCase.clearMockHistory`.

matlab.test.behavior.Missing class: Verify class satisfies missing-value behavior contract

Create a test class that derives from the `matlab.test.behavior.Missing` class to test if the missing value for the class satisfies the missing-value contract in MATLAB. If your class represents a

data type and you want MATLAB to treat missing values of your class similar to built-in classes, ensure that your class satisfies the missing-value contract.

Typically, you use the behavior test as part of a test-driven development workflow. If you want the missing value for your class to satisfy the missing contract with MATLAB, write the behavior test and modify the class under test until the test results are as you expect.

For more information, see `matlab.test.behavior.Missing`.

MEX Functions: Build Fortran MEX Files with Interleaved Complex API

The Fortran Matrix API supports the interleaved storage representation of complex numbers. For more information, see MATLAB Support for Interleaved Complex API in MEX Functions.

Note To run a Fortran MEX file built with the interleaved complex API in MATLAB R2018a, you must use MATLAB R2018a Update 3.

Compatibility Considerations

If you build Fortran MEX functions, then you should review the Do I Need to Upgrade My MEX Files to Use Interleaved Complex API? topic.

The functionality for several Fortran Matrix API functions has changed. For information, see:

- “Fortran Matrix API functions mxGetPi, mxSetPi, mxGetImagData, and mxSetImagData incompatible with interleaved complex API” on page 4-33
- “Change of behavior for Fortran Matrix API functions mxGetPr, mxSetPr, mxGetData, and mxSetData” on page 4-33
- “Change of behavior for Fortran Matrix API function mxGetElementSize” on page 4-33
- “Change of behavior for Fortran Matrix API functions mxCopyComplex16ToPtr, mxCopyPtrToComplex16, mxCopyComplex8ToPtr, and mxCopyPtrToComplex8” on page 4-33

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	MinGW®-w64 version 6.3.0 compiler from https://mingw-w64.org	Windows
Added	Intel Parallel Studio XE 2018 with Microsoft Visual Studio 2015 and 2017 for C, C++, and Fortran	Windows
Added	Intel Parallel Studio XE 2018 for Fortran	macOS
Discontinued	Microsoft Visual C++® 2013 Professional	Windows

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

System objects: Flexible requirements for inputs when calling System objects

In MATLAB, you can now call System objects with fewer inputs than those defined in the `stepImpl` or `outputImpl` methods. When the System object runs, the algorithm determines how inputs are used. This flexibility matches default MATLAB behavior for other functions and objects.

The System object algorithm might not be flexible for inputs depending on how the algorithm is implemented or if the System object implements the `getNumInputsImpl` method.

System object authoring: Use enumerations to define finite property lists in System objects

When adding a System object property with a finite list of values, use enumerations to define the allowed values. To add enumerations, open your System object file in the MATLAB editor, and select **Insert Property > Enumerations**.

For information about converting StringSets to enumerations, see “System object authoring StringSet class will be removed” on page 4-34.

Reference Architecture: Deploy and run MATLAB on Amazon Web Services (AWS) and Microsoft Azure

You can deploy and run the MATLAB desktop on AWS® and Azure® and connect to it using the Remote Desktop Protocol (RDP). Use a MathWorks provided AWS CloudFormation template to deploy to AWS and an Azure Resource Manager (ARM) template to deploy to Microsoft Azure. For more information, see MATLAB in the Cloud (https://www.mathworks.com/cloud.html). For reference architecture deployment details, see:

- MATLAB on AWS
- MATLAB on Azure

Git Stashes: Store uncommitted changes for later use

Starting in R2018b, you can use Git stashes directly from MATLAB. You can create a Git stash to store uncommitted changes for later use. For details, see Use Git Stashes.

Functionality being removed or changed

matlab.unittest.plugins.FailureDiagnosticsPlugin is not recommended
Still runs

The `matlab.unittest.plugins.FailureDiagnosticsPlugin` class is not recommended. Use the `matlab.unittest.plugins.DiagnosticsOutputPlugin` class instead. There are no plans to remove `FailureDiagnosticsPlugin` at this time.

Programmatic dependence on specific diagnostic subclass from getDiagnosticFor method of constraint and tolerances

Rework code that relies on properties or methods specific to `matlab.unittest.diagnostic.ConstraintDiagnostic` instances returned from the `getDiagnosticFor` method of `matlab.unittest.constraints` classes.

As of R2018b, diagnostics returned from constraint and tolerance classes in the `matlab.unittest.constraints` package are instances of `matlab.unittest.diagnostics.FrameworkDiagnostic`.

Protected access for matlab.unittest.fixtures.Fixture.onFailure
Behavior change in future release

The `matlab.unittest.fixtures.Fixture.onFailure` method will have protected access in a future release. Currently, the `Fixture.onFailure` method has public access. However, this method is designed to be used in subclasses of the `Fixture` class. In a future release, the `Fixture.onFailure` method will have protected access, which restricts use of the method to subclasses of the `matlab.unittest.fixtures.Fixture` class.

Fortran Matrix API functions mxGetPi, mxSetPi, mxGetImagData, and mxSetImagData incompatible with interleaved complex API
Still runs

Do not use `mxGetPi` and `mxSetPi` functions in Fortran MEX files built with the interleaved complex API (`mex` option `-R2018a`). Use `mxGetComplexDoubles (Fortran)` instead of `mxGetPr` and `mxGetPi`.

Do not use `mxGetImagData` and `mxSetImagData` functions in Fortran MEX files built with the interleaved complex API (`mex` option `-R2018a`). Use Typed Data Access functions instead.

For more information, see MATLAB Support for Interleaved Complex API in MEX Functions.

Change of behavior for Fortran Matrix API functions mxGetPr, mxSetPr, mxGetData, and mxSetData
Still runs

Do not use the `mxGetPr` and `mxSetPr` functions in Fortran MEX files built with the interleaved complex API (`mex` option `-R2018a`) for complex arrays. Use these functions for real arrays only, or use Typed Data Access functions.

Do not use the `mxGetData` and `mxSetData` functions in Fortran MEX files built with the interleaved complex API (`mex` option `-R2018a`) for numeric arrays. Use these functions for nonnumeric arrays only. For numeric arrays, use Typed Data Access functions.

Change of behavior for Fortran Matrix API function mxGetElementSize
Behavior change in future release

For a complex Fortran `mxArray` built with the interleaved complex API (`mex` option `-R2018a`), `mxGetElementSize (Fortran)` returns twice the value that the function in the separate complex API returns.

Change of behavior for Fortran Matrix API functions mxCopyComplex16ToPtr, mxCopyPtrToComplex16, mxCopyComplex8ToPtr, and mxCopyPtrToComplex8
Behavior change in future release

The function signatures for Fortran Matrix API functions `mxCopyComplex16ToPtr (Fortran)`, `mxCopyPtrToComplex16 (Fortran)`, `mxCopyComplex8ToPtr (Fortran)`, and

`mxCopyPtrToComplex8 (Fortran)` are different for MEX files built with the interleaved complex API (`mex` option `-R2018a`). The functions built with the separate complex API have two arguments for the data, the real and complex parts. The functions built with the interleaved complex API have a single argument for the data.

System object authoring StringSet class will be removed

Still runs

The class `matlab.system.StringSet` will be removed in a future release. To bring System object infrastructure closer to MATLAB classes, regular MATLAB enumerations replace the System object-specific StringSet functionality. To define a finite set of properties in System objects, use enumerations instead.

Update Code

StringSet Properties	Enumeration Properties

`properties`
`    Flavor = 'Chocolate'`
`end`
 	
`properties (Hidden,Constant)`
`    FlavorSet = matlab.system.StringSet...`
`       ({'Vanilla','Chocolate'})`
`end`
	In your System object class, define the property:
	
	`properties`
	`    Flavor (1,1) FlavorValues`
	`end`
	
	In a separate file, define the enumeration class:
	
	`classdef FlavorValues < int32`
	`    enumeration`
	`        Chocolate (0)`
	`        Vanilla (1)`
	`    end`
	`end`

In the MATLAB Editor, use the **Insert Property > Enumeration** for help in creating the enumeration class.

Compatibility Considerations

MATLAB enumerations do not support special characters or spaces for the enumeration values. When converting from StringSets to enumerations, remove or replace these characters in the enumeration class.

matlab.editor.autoformat.DoubleHashtagForHeading setting has been removed

Errors

The `matlab.editor.autoformat.DoubleHashtagForHeading` setting has been removed. Use the `matlab.editor.autoformat.HashtagsForHeading` setting instead.

To update your code, change instances of the setting `matlab.editor.autoformat.DoubleHashtagForHeading` to `matlab.editor.autoformat.HashtagsForHeading`. For more information, see matlab.editor Settings.

R2018a

Desktop

Live Editor: Create live functions with richly formatted documentation, including equations and images

In the Live Editor, you can create live functions that accept inputs and return outputs. To document your live functions, add richly formatted text, which includes equations, images, and formatted code examples. Then, you can use the `doc` command to view the documentation in the Help browser. For more information, see Create Live Functions.

Live Editor: Debug live functions and scripts

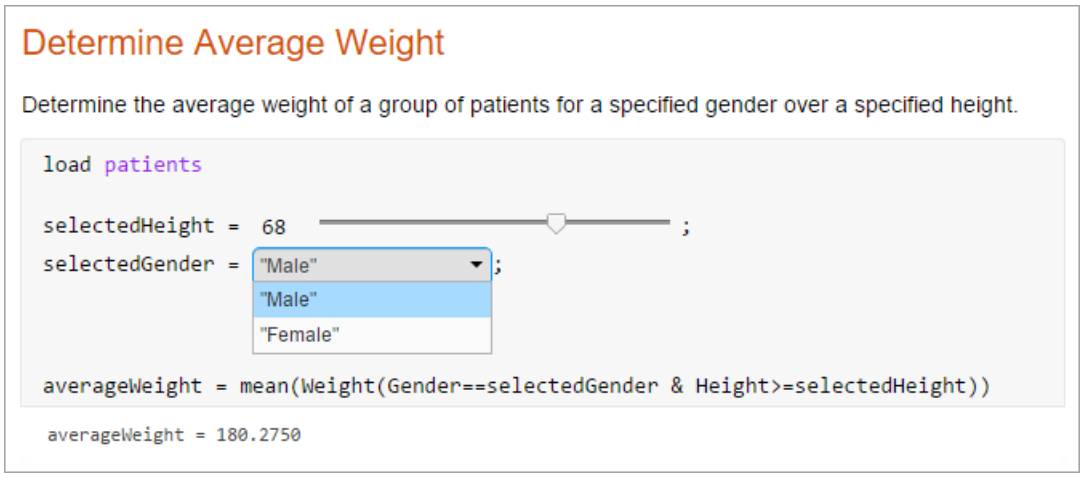
To diagnose problems in live functions and live scripts, debug your code in the Live Editor. You can use several methods to debug in the Live Editor:

- Show output by removing semicolons.
- Run to a specific line of code and pause using the button.
- Investigate called functions and scripts by stepping in using the button.
- Add breakpoints to your file to enable pausing at specific lines when you run the file.

For more information, see Debug Code in the Live Editor.

Live Editor: Add sliders and drop-down lists to control variable values in a live script

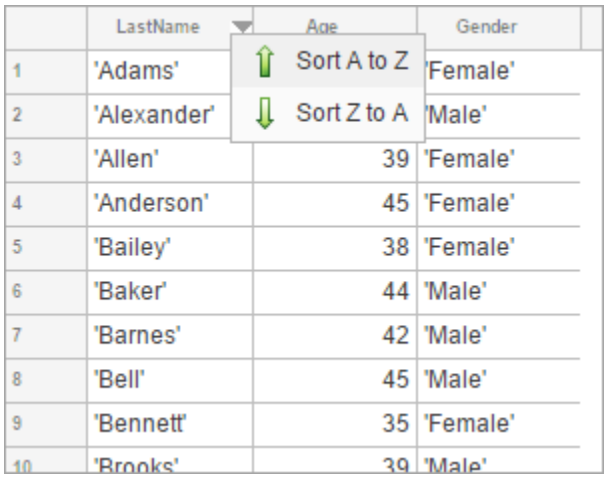
You can add sliders and drop-down lists to your live scripts to interactively control variable values. Adding interactive controls to a live script is useful when you want to share the live script with others. Use the controls to define and limit the values of variables that others can change in your live script.



To add a numeric slider or drop-down list, go to the **Insert** tab, click **Control**, and select from the available options. For more information, see Add Interactive Controls to a Live Script.

Live Editor: Sort table data interactively

In the Live Editor, you can sort table data interactively. To sort data in a table, click the down arrow ▾ to the right of a variable name in the table and select from the available sorting options.



Use the **Update Code** button below the table to add the generated code to your live script. Adding the generated code to your live script ensures that the sorting is reproduced the next time you run the live script.

Live Editor: Create a table of contents and add formatted code examples

You can create tables of contents in live scripts and functions that contain a list of all the titles and headings in the document. To insert a table of contents, go to the **Insert** tab and select **Table of Contents**. Only the title of the table of contents is editable.

You also can add formatted code examples to live scripts and functions. A code example is sample code that appears as text. To add a code example, go to the **Insert** tab, click **Code Example** and select **Plain**. The Live Editor displays the sample code as indented and monospaced text. To add a MATLAB syntax highlighted code example, go to the **Insert** tab, click **Code Example** and select **MATLAB**.

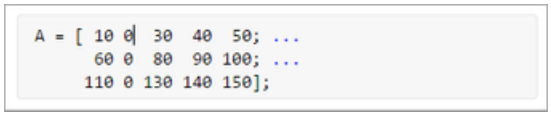
For more information about adding formatted text to live scripts and functions, see Format Files in the Live Editor.

Live Editor: Select and edit a rectangular area of code

In the Live Editor, you can select a rectangular area in your code (also known as column selection or block edit) by pressing the **Alt** key while making a selection. Selecting and editing a rectangular area of code is useful if you want to copy or delete several columns of data, or if you want to edit multiple lines at one time. For example, select the second column of data in A.

```
A = [ 10  20  30  40  50; ...
      60  70  80  90 100; ...
     110 120 130 140 150];
```

Type `0` to set all the selected values to `0`.



Add-Ons Explorer: Browse by category to discover convenient, helpful add-ons

Use the available categories to browse for new, convenient, and helpful add-ons in the Add-Ons Explorer. For more information about add-ons and how to find and install them, see Get Add-Ons.

Comparison Tool: Find differences in live scripts and functions

Use the Comparison tool to find differences in live scripts and functions. The Comparison tool highlights differences in both the code and the text.

To start a comparison of a live script or function, go to the **Live Editor** or **Home** tab, click **Compare**, and then select the files you want to compare. To start a comparison from the Current Folder browser, select a file, right-click, and select **Compare Against**.

Favorites: Rerun favorite commands

Create favorite commands (previously called command shortcuts) to easily rerun a group of MATLAB language statements that you use regularly. To access existing favorite commands, go to the **Home** tab and in the **Code** section, click **Favorites** and then select from available favorite commands. To create a new favorite command, click **Favorites** and then press the **New Favorite** button. For more information, see Rerun Favorite Commands.

Toolbox Packaging: Specify portability information for custom toolboxes

You can indicate which platforms and MATLAB releases support your custom toolbox. When someone installs your toolbox on an unsupported platform or MATLAB release, MATLAB displays a warning. However, they can still install the toolbox. For more information, see Create and Share Toolboxes.

Language and Programming

Empty Arrays: Create complex empty arrays using functions such as zeros and ones

You can create complex empty arrays using the `complex` function and functions that support complex input with the `'like'` syntax, such as `zeros` and `ones`. For example, the commands `zeros(0,0,'like',1i)` and `complex([])` now return complex empty output instead of real empty output.

Compatibility Considerations

Previously, the `complex` function and functions that support complex input with the `'like'` syntax always returned real empty arrays when one or more dimensions were 0. To preserve this behavior, you can check for complex empty arrays using the `isreal` function and convert them to real empty arrays using the `real` function.

Code Compatibility Report: Generate compatibility report from Current Folder browser

The Code Compatibility Report displays potential compatibility issues in your code and categorizes them by severity. This report helps you to update code to a newer MATLAB release by grouping issues into those issues that require immediate resolution and those issues you can address in a later release.

As of R2018a, you can generate the Code Compatibility Report from the Current Folder browser. Previously, you generated the report using the `codeCompatibilityReport` function only. For more information, see MATLAB Code Compatibility Report.

timer Object: Access properties with multilevel indexing

The `timer` object supports multilevel indexing for setting and accessing properties. For example, consider a timer `t` with a `UserData` property that contains a structure. Now you can access the data in the structure directly.

```
t.UserData.field1
```

To access the `field1` value in previous versions of MATLAB, you either defined an intermediate variable, or used the `getfield` function.

```
d = t.UserData;
d.field1

getfield(t.UserData, 'field1')
```

Now you can set the values in the structure directly.

```
t = timer;
t.UserData.field1 = 'value1';
t.UserData.field2 = 'value2';
```

Using dot notation to access properties on an array of `timer` objects now returns a comma-separated list instead of a cell array. Using dot notation to set properties on an array of `timer` objects now errors.

Compatibility Considerations

If your code relies on MATLAB returning a cell array when getting the properties for an array of `timer` objects, either update your code to work with a comma-separated list, or use the `get` function. This code continues to return a cell array.

```
t_arr = [timer timer timer];
get(t_arr,'Name')

ans =

  3×1 cell array

    {'timer-5'}
    {'timer-6'}
    {'timer-7'}
```

To continue to set properties for all elements in an array of `timer` objects, use the `set` function.

```
set(t_arr,'TimerFcn','myFunction')
```

In MATLAB R2018a and later, using dot notation to set the properties for all `timer` objects in an array results in an error. For example, `t_arr.TimerFcn = 'myFunction'` now errors.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`complex` function and functions supporting complex input with the `'like'` syntax	Still runs	Not applicable	Previously, the `complex` function and functions that support complex input with the `'like'` syntax (such as `zeros` and `ones`) always returned real empty arrays when one or more dimensions were 0. To preserve this behavior, you can check for complex empty arrays using the `isreal` function and convert them to real empty arrays using the `real` function.

Functionality	Result	Use This Instead	Compatibility Considerations
MATLAB path names containing `'.'`, `'..'`, and symbolic links	Still runs	Not applicable	MATLAB now resolves all path names containing `'.'`, `'..'`, and symbolic links to their target location before adding them or removing them from the path. Resolving the path names ensures that each entry in the MATLAB path represents a unique folder location. For example, if you run the command `addpath('c:\matlab\..\work')`, MATLAB adds the folder `c:\work` to the path. MATLAB also resolves path names when changing the current folder. For more information, see `addpath`, `rmpath`, `path`, and `cd`.
Case sensitivity when adding or removing folders to the MATLAB path.	Still runs	Not applicable	MATLAB now case corrects path names according to the file system's case characteristics before adding or removing them from the path. For example, if the folder `c:\TEMP` exists on your system and you run the command `addpath('c:\temp')`, MATLAB adds the folder `'c:\TEMP'` to the path. For UNC paths, MATLAB case corrects only the file path part of the path. The server name and share name parts of the path are standardized to all lowercase.

Functionality	Result	Use This Instead	Compatibility Considerations

`ans` in nested functions	Still runs	Not Applicable	Currently, nested functions sometimes can access and modify the `ans` variable that is defined in the parent function. In future releases, nested functions will never have access to the `ans` variable defined in the parent function. For example in future releases, in this code, MATLAB throws an error in the nested function.
			`function foo`
			`    1+2 % Assigns to ans.`
			`    nested()`
			`    function nested`
			`       ans % MATLAB throws an error.`
			`    end`
			`end`
			To share a value between a nested function and its parent function, assign the value to a variable other than `ans`.
			`function foo`
			`    sharedVar = 1+2`
			`    nested()`
			`    function nested`
			`       sharedVar`
			`    end`
			`end`

Functionality	Result	Use This Instead	Compatibility Considerations

MATLAB classification of identifiers in anonymous functions	Still runs	Not applicable	Currently, if any identifiers in an anonymous function are unknown at creation time, all identifiers are unknown. In a future release, they will be classified using the same rules as if they were inside the function. If an identifier is known in the function, it is known in the anonymous function. For example, consider the following function and its local function.
			
			`function myfun`
			`  myscript; % script sets x = 1 and lf = 10`
			`  f1 = @()lf(1);`
			`  f2 = @()lf(x);`
			`end`
			`function lf(y)`
			`  % local function to myfun`
			`end`
			
			All identifiers in `f1` are known when MATLAB creates the anonymous function. It calls the local function `lf` with an input of 1. In the current version of MATLAB, since `x` is unknown, MATLAB considers `lf` to be unknown also, and resolves both identifiers at runtime. Therefore, it uses `x` from `myscript` to index into the variable `lf` from `myscript`.
			
			In a future version of MATLAB, `lf` in `f2` will resolve to the local function, since `lf` is known when the anonymous function is created. `x` continues to be unknown until runtime. At runtime, MATLAB will use `x` from `myscript` as input to the local function `lf`, instead of as an index to the variable `lf` from `myscript`.

Functionality	Result	Use This Instead	Compatibility Considerations

MATLAB resolution of identifiers within `parfor`/`spmd` blocks	Still runs	Not applicable	Currently, MATLAB resolves identifiers in `parfor`/`spmd` blocks only within the scope of that block. In a future version of MATLAB, these identifiers will have the same resolution outside the `parfor`/`spmd` block. For example, consider the following function.
			`function myFunc()`
			`  parfor i = 1:2`
			`  y(i);`
			`  end`
			`  i`
			`end`
			Currently, inside the `parfor` block, `i` is recognized as a variable, but outside of the `parfor` block, it is an unrecognized function or variable. In a future version of MATLAB, `i` will be recognized as a variable outside of the `parfor` loop. However, its value from the `parfor` loop is not available outside of the block. In this example, the value of `i` is `sqrt(-1)` outside of the `parfor` loop.

Change in precedence of compound name resolution	Still runs	Not applicable	In MATLAB, a compound name is a name comprised of several parts joined by a dot. For example `structName.fieldName` or `packageName.ClassName.methodName`. In the current version of MATLAB, if a compound name, such as `a.b.c`, does not resolve to a variable, then it has the following precedence order. **1** Class called `a.b.c` **2** Class called `a.b` with static method `c` **3** Package function called `a.b.c` **4** Class called `a.b` (no static method `c`) **5** Class called a In a future version of MATLAB, if `a.b.c` does not resolve to a variable, it will have this precedence order. **1** Class or package function called `a.b.c` **2** Class or package function called `a.b` **3** Class or function called `a`
Deleting a folder that was removed outside of MATLAB	Warns	Not applicable	MATLAB now detects and warns if you try to delete a folder that was removed outside of MATLAB.

Functionality	Result	Use This Instead	Compatibility Considerations
notebook	Errors	Live Editor	`notebook` has been removed. To create a document that combines code, formatted text, and output, use the Live Editor instead.

Change in precedence of *based imports	Warns	To use a package function, use the fully qualified name.	Imports in a function have the highest precedence. Imports shadow variables, local functions, and nested functions. For example,
			`function myfunc`
			`%import "local" and "nest" functions`
			`  import pkg1.*`
			`  local()  % Calls "pkg1.local"`
			`           % and displays warning.`
			
			`  function nest`
			`  end`
			
			`  nest(); % Calls "pkg1.nest"`
			`          % and displays warning.`
			`end`
			
			`function local`
			`end`
			
			In the future, imported functions will have lower precedence than variables, nested functions, and local functions
			
			`function myfunc`
			`%import "local" and "nest" functions`
			`  import pkg1.*`
			`  local()  % Calls "myfunc/local"`
			
			`  function nest`
			`  end`
			
			`  nest(); % Calls "myfunc/nest"`
			`end`
			
			`function local`
			`end`

Functionality	Result	Use This Instead	Compatibility Considerations

Fully qualified import behavior	Warns	To use a function "nest" from a fully qualified import instead of the nested function with the same name, either remove the import statement or the nested function definition.	A fully qualified import takes precedence over a nested function.
			
			`function myfunc`
			`% import function "nest"`
			`  import pkg.nest`
			`% Calls "pkg.nest" and displays warning`
			`  nest();`
			
			`  function nest`
			`  end`
			`end`
			
			`myfunc`
			
			`Warning: Prec-1.File: myfunc.m Line: 2 Column: 10`
			` Using "nest" from the fully qualified`
			`import instead of the nested function`
			`with the same name.`
			` This will error in a future release.`
			`Either remove the import statement or`
			` the nested function definition.`
			
			In the future, fully qualified imports sharing names with identifiers in the same scope will throw an error.

Functionality	Result	Use This Instead	Compatibility Considerations

Fully qualified imports with names that shadow identifiers in outer scopes	Warns	Not applicable	A fully qualified import is ignored when it shadows an identifier in the outer scope.
			
			`1.function myfunc`
			`2.  x = 1;`
			`3.  function nest`
			`4.    import pkg1.x`
			`% Calls variable "myfunc/x" on line 2`
			`% and displays warning`
			`5.    x()`
			`6.  end`
			`7.end`
			
			`myfunc`
			
			`Warning`
			`ID: mir_nr_id_shadows_imported_function`
			`Message: "x" is both the name of an imported`
			`function in a nested function and`
			`the name of a variable in the outer scope.`
			`The imported "x" function is not called.`
			`In future, the imported "x" will be called.`
			
			In the future, fully qualified imports will shadow identifiers in the outer scopes with same name.
			
			`1.function myfunc`
			`2.  x = 1;`
			`3.  function nest`
			`4.    import pkg1.x`
			`5.    x() % Calls imported function "pkg1.x"`
			`6.  end`
			`7.end`

Error handling for fully qualified imports that cannot be resolved	Still runs	Not applicable	If MATLAB is launched without Java, then fully qualified imports that cannot be resolved do not throw an error. If MATLAB is launched with Java, then fully qualified imports that cannot be resolved throw an error. In the future, fully qualified imports that cannot be resolved throw an error with or without Java.

Functionality	Result	Use This Instead	Compatibility Considerations
Imports used in scripts compared with imports used in functions	Still runs	Not applicable	If a script contains an import statement and that import statement matches the name of a class on the path, then all uses of that identifier resolve to the class. If the identifier does not resolve to a class at runtime, then an error is thrown. In the future, imports used in scripts will behave like imports in functions.

Error handling when import not found	Still runs	Remove functions such as `javachk` and `usejava` used for error handling.	Fully qualified imports that cannot be resolved throw an error in MATLAB `-nojvm` mode. For example, create this function.
			
			`function myfunc`
			`  import java.lang.String`
			`  if ~usejava('jvm')`
			`    disp('This function requires Java');`
			`  else`
			`    % do something with java String class`
			`  end`
			`end`
			
			Start `matlab -nojvm`, and then run the function.
			
			`myfunc`
			
			`This function requires Java`
			
			In the future, MATLAB throws an error. For example:
			
			`myfunc`
			
			`Unable to find class or function 'java.lang.String`
			`(line 2) for import. If your class or function`
			`requires Java, restart MATLAB without the`
			`-nojvm option.`

Mathematics

graph and digraph Objects: Work with multigraphs that have multiple edges between two nodes

The `graph` and `digraph` classes now support multiple edges between two nodes. Repeated edges are distinct and can have different weights. To facilitate working with graphs that have repeated edges, several new functions are available:

- `ismultigraph` — Determine whether a graph is a multigraph or a simple graph.
- `simplify` — Reduce a multigraph into a simple graph. Optionally, you can specify a rule to combine or pick between repeated edges.
- `edgecount` — Count the number of edges between two nodes.
- `outedges`, `inedges` — Find outgoing or incoming edge indices for a particular node.

Additionally, some existing graph functions have updated capabilities to account for multiple edges.

Function	New Functionality
`shortestpath`	New third output that lists the edge indices of all edges on the path.
`shortestpathtree`	New third output that indicates whether each edge is in the tree.
`highlight`	All edges between the source and target nodes are highlighted by default. A new name-value pair `'Edges'` enables you to highlight specific edges using indices, which is compatible with the new third output of `shortestpath`.
`bfsearch`, `dfsearch`	New second output that contains the edge index. If the primary output is a table, then it has an additional variable `EdgeIndex` with the same information.
`isomorphism`	New second output that contains permutation information of the edges.
`findedge`	New second output that contains edge indices.

Compatibility Considerations

There are a few changes that might require updates to existing code:

1. **`neighbors` function counts self-loops only once.** In previous releases, if node `u` had a self-loop, then `neighbors(g,u)` listed `u` twice in the output. `neighbors(g,u)` now returns only one instance of `u`.
2. **Display change of self-loops in plot of a simple graph.** Self-loops in the plot of a simple graph are now shaped like a leaf or teardrop. In previous releases, self-loops were displayed as circles.
3. **`graph`, `digraph`, and `addedge` no longer produce errors when they encounter duplicate edges.** Instead, the duplicate edges are added to the graph and the result is a multigraph. The

`ismultigraph` function is useful to detect this situation, and `simplify` provides an easy way to remove the extra edges.

graph and digraph Objects: Calculate component sizes and weighted adjacency matrices

The `conncomp` function offers a second output to return the number of nodes in each connected component. Additionally, the `adjacency` function accepts a second input to create weighted adjacency matrices.

GraphPlot Object: Visualize graphs with additional options for 'force', 'force3', and 'circle' layouts

When you plot a graph `G` with a specific layout using `plot(G,'Layout',method)`, you can use a number of layout-specific name-value pairs for each of the different methods. For example, `plot(G,'Layout','layered','Direction','left')` changes the orientation of a layered layout.

New layout-specific name-value pairs for visualizing directed and undirected graphs with the `plot` function include:

- `'WeightEffect'` name-value pair — To incorporate edge weights into the `'force'` and `'force3'` layouts.
- `'UseGravity'` name-value pair — To turn the effects of gravity on (set to off by default) with the `'force'` and `'force3'` layouts. When gravity is turned on, plotting a graph with multiple components attracts all of the nodes to the origin and enables large components to take up more space.
- `'Center'` name-value pair — To specify a single node that should appear in the center of a circular graph plot with the `'circle'` layout.

polyshape Objects: Analyze polygons with turningdist, nearestvertex, and overlaps functions

New functionality for analyzing 2-D polygons is available for `polyshape` objects:

- `turningdist` function returns a number close to 0 when two input `polyshape` objects have nearly matching boundary shapes, regardless of scale or orientation.
- `nearestvertex` function returns the closest vertex of an input `polyshape` object to a query point.
- `overlaps` function determines if two `polyshape` objects are overlapping.

polyshape Objects: Return vertex map and accept arrays with compatible sizes for intersect, subtract, union, and xor functions

The `intersect`, `subtract`, `union`, and `xor` functions for `polyshape` objects now return vertex mapping information and accept input arrays of `polyshape` objects with compatible sizes.

Vertex mapping enables you to identify where the vertices of an output `polyshape` originated. For example, `[pout,pshapeID,vertexID] = intersect(pshape1,pshape2)` returns column

vectors `pshapeID` and `vertexID` containing the same number of rows as the number of vertices in `pout`.

Each element of `shapeID` contains the value 1, 2, or 0:

- Element is 1 when the corresponding vertices of `pout` belong to `pshape1`.
- Element is 2 when the corresponding vertices of `pout` belong to `pshape2`.
- Element is 0 when the corresponding vertices of `pout` belong to the intersection of `pshape1` and `pshape2`.

The vertex information `vertexID` contains the row numbers for the corresponding vertices in `pshape1` and `pshape2`.

polybuffer Function: Create buffer around points or lines

The `polybuffer` function enables you to specify a buffer around a set of points or line segments. For example, `pshape = polybuffer([0 0; 2 3],'Points',1)` creates a `polyshape` object `pshape` whose boundaries are circles of radius 1 centered about the points (0,0) and (2,3).

triangulation Objects: Find neighboring vertices and locations of query points with improved performance

You can compute neighboring vertices and locations of query points faster with the `nearestNeighbor` and `pointLocation` functions for `triangulation` objects.

ode45 Function: Solve nonstiff differential equations faster

The `ode45` function shows improved performance for some problems.

Graphics

Axes Object: View axes at small size with improved layout, limit selection, and font scaling

Small axes have an improved layout that reduces white space for better readability. To reduce white space, the axes limits now fit more tightly around the data. Also, legends and colorbars have narrower margins. Additionally, the font size scales down slightly for small axes to avoid overlapping text or text that runs off the figure.

Compatibility Considerations

For small axes, you might encounter slightly different limits, tick values, or font sizes than in previous releases. However, if you previously specified the limits, tick values, or font size, then the axes still uses the values that you specify. To specify the limits, use the `xlim` and `ylim` functions. To specify the tick values, use the `xticks` and `yticks` functions. To specify the font size, use the `FontSize` property.

Axes Object: Map data values to colormap using linear or logarithmic scale

Scale colormaps linearly or logarithmically using the new ColorScale property for `Axes` objects.

Legend Object: Create legends with multiple columns

You can create legends with multiple columns using the new `NumColumns` property for `Legend` objects. For an example, see Add Legend to Graph.

heatmap Function: Zoom and pan data, display data tips, and sort rows and columns interactively

Heatmaps have new options for interacting with data:

- Zoom — Use the scroll wheel or the **+** and **-** keys to zoom.
- Pan — Use the arrow keys to pan across the rows or columns.
- Data tips — Hover over the heatmap to display a data tip.
- Rearrange rows and columns — Click and drag a row or column label to move it to a different position.
- Sort values — Click the icon that appears when you hover over the row or column label. Click once to sort the values in ascending order, twice to sort the values in descending order, and a third time to reset the order.

geobubble Function: Explore with interactive data tips and a scale bar

Geographic bubble charts now support data tips. Hover over a bubble on the chart to display a data tip.

In addition, geographic bubble charts now include a scale bar in their lower left corner to indicate the actual distances represented in the map. Use the `ScalebarVisible` property to control whether your chart includes a scale bar.

Axes Toolbar: Add toolbars to your axes for quick access to pan, zoom, and other data exploration tools

You can add a toolbar to the top-right corner of the axes for quick access to data exploration tools. The toolbar typically includes options to brush data, add data tips, rotate (3-D axes only), pan, and zoom.

To add the toolbar, use the new `Toolbar` property for `Axes` objects. The `Toolbar` property stores an `AxesToolbar` object. Set the `Visible` property of the `AxesToolbar` object to `'on'`. Some of the axes toolbar options are redundant with the figure toolbar. You can remove the redundant options from the figure toolbar using the `removeToolbarExplorationButtons` command.

```
ax = gca;
ax.Toolbar.Visible = 'on';
removeToolbarExplorationButtons(gcf);
```

To restore the figure toolbar exploration buttons, use the `addToolbarExplorationButtons(gcf)` command.

Property Inspector: Modify graphics interactively with an improved property inspector

An improved **Property Inspector** makes it easier to modify graphics objects interactively. To open the inspector, use the `inspect` function. For example, `inspect(gca)` opens the inspector for the current axes.

Polygon Object: Control color and transparency of hole edges

You can control the color and transparency of the polygon hole boundaries. Use the new `HoleEdgeColor` and `HoleEdgeAlpha` properties for `Polygon` objects.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Changing the figure colormap affects all axes in the figure	Still runs	Not applicable	When you change a figure's colormap, the colormap changes for all axes in the figure. Previously, if you specified the colormap for a particular axes, then changing the figure's colormap did not affect the axes. Set the axes colormap after setting the figure colormap to avoid the figure's colormap overwriting the axes colormap.
Axes limits, tick values, or font sizes for small axes might be different	Still runs	Not applicable	Due to an improved layout that reduces white space, small axes might have slightly different limits, tick values, or font sizes than in previous releases. If you prefer the limits, tick values, or font sizes from previous releases, you can manually set them. To specify the limits, use the `xlim` and `ylim` functions. To specify the tick values, use the `xticks` and `yticks` functions. To specify the font size, use the `FontSize` property.
Setting title properties of geographic bubble chart to empty (`[]`) array	Still runs	Not applicable	GeographicBubbleChart objects now store empty title properties as an 0-by-0 empty character array. The title properties include the `Title`, `SizeLegendTitle`, and `ColorLegendTitle` properties. Previously, setting the property to an empty array either returned an error or stored the value with unexpected dimensions (as a 0-by-0-by-0 empty character array).
Setting title properties of geographic bubble chart to multiline title	Still runs	Not applicable	GeographicBubbleChart objects now store multiline titles as a cell array of character vectors. The title properties include the `Title`, `SizeLegendTitle`, and `ColorLegendTitle` properties. Previously, the chart stored the value as a character matrix and displayed the title with incorrect alignment.

Data Import and Export

readtable Function: Specify the number of rows to read from a text file using import options

You can select a subset of rows to read from a text file using `readtable` along with import options. Use the `DataLines` property to specify the rows to read. Specify `DataLines` in multiple ways:

- `opts.DataLines = 5` sets the `DataLines` property to the value `[5 inf]`. The `readtable` function reads all rows of data starting from row `5` to the end-of-file.
- `opts.DataLines = [2 6]` sets the property to read lines `2` through `6`. The `readtable` function reads all rows of data starting from row `2` to row `6`.
- `opts.DataLines = [1 3; 5 6; 8 inf]` sets the property to read rows `1`, `2`, `3`, `5`, `6`, and all rows between `8` and the end-of-file.

After specifying `DataLines`, import the data using `readtable`:

```
T = readtable(filename,opts);
```

For more information, see the `DataLines` property of `DelimitedTextImportOptions` and `FixedWidthImportOptions` objects.

readtable Function: Easily manage prefixes and suffixes from data using import options

You can specify characters to remove from the prefix and suffix positions in a variable using `readtable` with import options. Specify the prefix and suffix options in multiple ways:

- `opts = setvaropts(opts,'Var1','Prefixes','$')` sets the `Prefixes` option for the variable `Var1`. If `Var1` contains a value of `'$500'`, then `readtable` reads it as `'500'`.
- `opts = setvaropts(opts,'Var1','Suffixes','/-')` sets the `Suffixes` option for the variable `Var1`. If `Var1` contains a value of `'$500/-'` , then `readtable` reads it as `'$500'`.
- `opts = setvaropts(opts,'Var1','TrimNonNumeric',true)` sets the `TrimNonNumeric` option for variable `Var1`. If `Var1` contains a value of `'$500/-'`, then `readtable` reads it as `500`.

For more information, see the variable options and descriptions on the `setvaropts` reference page.

preview Function: Preview first 8 rows of a table in a file without importing the full table

You can preview the first `8` rows of a table from a file before importing the full table using `readtable` and import options. For more information, see the `preview` function.

imageDatastore Function: Work with millions of images with improved memory usage and performance

The `imageDatastore` function now supports faster processing of millions of images for deep learning applications with improved memory usage.

Datastore Functions: Seamlessly work with datasets stored on cloud and local machines

You can work with datasets stored on cloud and local machines by using the `'AlternateFileSystemRoots'` parameter in datastore functions and the File-set object for custom datastores. This parameter is supported for:

- Datastore objects: `TabularTextDatastore`, `SpreadsheetDatastore`, `ImageDatastore`, `FileDatastore`, `KeyValueDatastore`, and `TallDatastore`.
- File-set object: `matlab.io.datastore.DsFileSet` for custom datastores.

For more information, see Set Up Datastore for Processing on Different Machines or Clusters.

Datastore Functions: Read HDFS data more easily when using Hortonworks or Cloudera

You can use datastore functions to access HDFS data more easily on Hortonworks or Cloudera® without having to set the `HADOOP_HOME` or `HADOOP_PREFIX` environment variables. MATLAB now automatically assigns these environment variables when using Hortonworks or Cloudera application edge nodes.

For more information, see Read Remote Data.

readtable, detectImportOptions, datastore, and tabularTextDatastore Functions: Automatically detect and return duration data in text files

The functions `readtable`, `detectImportOptions`, `datastore`, and `tabularTextDatastore` detect and return duration data as `duration` type. For more information on `duration` arrays, see the `duration` reference page.

Compatibility Considerations

Previously, `readtable`, `detectImportOptions`, `datastore`, and `tabularTextDatastore` functions returned duration data as text. To preserve that behavior, use the `DurationType` parameter. For example, `T = readtable('myfile.txt','DurationType','text')` reads duration data in `myfile.txt` as text. For more information on the `DurationType` parameter, see the function reference pages.

detectImportOptions Function: Control import properties of duration data

The `detectImportOptions` now can detect and manage variables of type `duration` in tabular data.

For a variable in a table containing duration data, you can use the `setvaropts` function to set these properties: `DurationFormat`, `InputFormat`, `DecimalSeparator`, and `FieldSeparator`. For more information, see the `setvaropts` function page.

VideoReader Function: Read video files faster on all platforms

`VideoReader` has improved performance for all supported video formats when you read video frames in a loop as part of a general video processing workflow. For more information on supported video formats and function usage, see `VideoReader`.

VideoWriter Function: Write video files faster on all platforms

`VideoWriter` has improved performance for all supported video formats, when you write video frames in a loop as part of a general video processing workflow. For more information on supported video formats and function usage, see `VideoWriter`.

openDiskFile Function: Read data files in FITS (Flexible Image Transport System) data format

Read data files in FITS (Flexible Image Transport System) data format using the `openDiskFile` function.

webwrite Function: Support for NTLM authentication

The `webwrite` function and `weboptions` POST and PUT methods support NTLM authentication on Windows platforms.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`DataLine` property of `DelimitedTextImportOptions` and `FixedWidthImportOptions` objects	Still runs	`DataLines`	Use property name `DataLines` instead of `DataLine`.
`readall` method for `SpreadsheetDatastore`, `TabularTextDatastore`, `KeyValueDatastore`, and `TallDatastore` objects	Still runs	Not applicable	The `readall(ds)` method no longer resets the datastore `ds` to a state where no data has been read from it. This new behavior of `readall` enables the `read` method to read data from the end of the previous read operation. Previously, `readall(ds)` would reset the datastore specified by `ds` to the state where no data has been read from it. Therefore, any call to `read` after a `readall` would always read from the beginning of the datastore.

Functionality	Result	Use This Instead	Compatibility Considerations

Output from `sheetnames` function	Still runs	Not applicable	The `sheetnames` function now returns the `names` argument as a string array.
			
			Previously, `sheetnames` returned `names` as a cell array of character vectors. To convert `names` back to a cell array of character vectors, use the following:
			
			`% Get sheet names from a file 'myfile3.xls'`
			`% in the spreadsheetDatastore ssds`
			`names = sheetnames(ssds,'myfile3.xls');`
			`% Convert names to cell array of character vectors`
			`names = cellstr(names);`

`hdfgd`	Errors	`matlab.io.hdfeos.gd`	Replace all instances of `hdfgd` with the corresponding function in the `matlab.io.hdfeos.gd` package.
`hdfsd`	Errors	`matlab.io.hdf4.sd`	Replace all instances of `hdfsd` with the corresponding function in the `matlab.io.hdf4.sd` package.
`hdfsw`	Errors	`matlab.io.hdfeos.sw`	Replace all instances of `hdfsw` with the corresponding function in the `matlab.io.hdfeos.sw` package.

Data Analysis

groupsummary Function: Group and discretize data for summary operations on table and timetable variables

Grouping and discretizing table and timetable data for summary operations is now easier with the `groupsummary` function. For example, `groupsummary(T,'Gender','method','mean')` computes the mean of the variables in a table `T` by gender.

Table and Timetable Variables: Add, delete, and rearrange column-oriented variables with the functions addvars, removevars, movevars, splitvars, mergevars, rows2vars, and inner2outer

Function	Description
`addvars`	Add variables to table or timetable.
`removevars`	Delete variables from table or timetable.
`movevars`	Move variables in table or timetable.
`splitvars`	Split multicolumn variables in table or timetable.
`mergevars`	Combine table or timetable variables into multicolumn variable.
`rows2vars`	Reorient rows of table or timetable so that rows become variables.
`inner2outer`	Invert a nested table-in-table hierarchy.

Preallocated Tables and Timetables: Initialize table and timetable variables so that they have specified sizes and data types

To initialize table and timetable variables so that they have specified sizes and data types, specify the `'Size'` and `'VariableTypes'` name-value pair arguments of the `table` and `timetable` functions. You can specify the number of rows, the number of variables, and the data types of the variables. The preallocated variables contain default values or empty arrays, depending on the specified data types. You can fill the variables with data values later.

Regular Timetables: Create regularly spaced timetables using a time step or sampling rate

You can create regular timetables using either a time step or a sampling rate to specify identical time intervals between consecutive row times. To create regular timetables, use the `timetable` or `array2timetable` functions with the `'TimeStep'` or `'SamplingRate'` name-value pair arguments. You also can specify the value of the first row time using the `'StartTime'` name-value pair.

retime and synchronize Functions: Synchronize timetables to a time step or sampling rate that you specify

To synchronize timetables to a time step or sampling rate, use the `'TimeStep'` or `'SamplingRate'` name-value pair arguments of the `retime` or `synchronize` functions.

duration Arrays: Create duration arrays from text that represents elapsed times

You can convert text representing elapsed times into a duration array using the `duration` function. The input text represents each time in a format such as `'hh:mm:ss'` or `'dd:hh:mm:ss'`. The fields `dd`, `hh`, `mm`, and `ss` represent days, hours, minutes, and seconds, respectively.

normalize Function: Normalize array, table, and timetable data

You can normalize data in an array, table, or timetable by quantities such as the z-score or p-norm using the `normalize` function. For example, `normalize(A,'norm',1)` normalizes each column of a matrix `A` by its 1-norm.

tall Arrays: Operate on tall arrays with more functions, including smoothdata, find, and isoutlier

The functions listed in this table add support for tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For more information on usage and limitations, type `help tall/functionName`. For example, `help tall/find`.

`array2timetable`	`isoutlier`
`caldiff`	`isprotected`
`filloutliers`	`maxk`
`find`	`mink`
`flip`	`rescale`
`fliplr`	`smoothdata`
`iscategory`	`splitlines`
`isordinal`	

tall Array Indexing: Use tall numeric arrays to index the first dimension

For a tall array `X`, you can index the first dimension of the array with `X(idx,…)`, where `idx` is a tall numeric array.

tall Arrays: Solve linear systems Ax = b

You can solve linear systems $Ax = b$ with a tall coefficient matrix `A` by `x = A\b`. The solver for tall arrays uses a QR decomposition to find a least-squares solution to the problem.

tall Arrays: Return group labels with findgroups

`findgroups` now supports multiple output arguments with tall arrays. The additional outputs from `findgroups` contain unique lists of group labels for each grouping variable.

tall Arrays: Set date and time components of tall datetime and tall duration arrays

You can change the properties of tall datetime and tall duration arrays using dot indexing. For example, `t.Format = 'dd-MMM-yyyy'` changes the display format of `t`.

See the `datetime` and `duration` pages for a list of the properties, or Extract or Assign Date and Time Components of Datetime Array for examples.

tall Arrays: Set properties of tall tables and tall timetables

You can change the properties of tall tables and tall timetables using dot indexing. For example, `T.Properties.VariableNames = {'Name1' 'Name2'}` changes the variable names of `T`. For a list of all properties use the command `T.Properties`.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations

Adding and subtracting durations represented as text in timer format from a datetime array	Still runs	Not applicable	Starting in R2018a, if you represent a duration as text, you can add and subtract it from a datetime array, and the result is a datetime array. In both operations, text in timer format is converted to a duration value. For example, this code adds 1 hour and 30 minutes to the current date and time.
			`datetime('now') + '1:30:00'`
			This code subtracts 12 hours from the current date and time.
			`datetime('now') - '12:00:00'`
			Previously, if you represented a duration as text in a timer format, such as `'1:30:00'`, then adding that duration to a datetime array raised an error, while subtracting it returned a duration array, instead of a datetime array.
			To reproduce the earlier behavior, convert the text to a datetime value. Subtracting a datetime array from another datetime array returns a duration array.
			`datetime('now') - datetime('12:00:00')`

Functionality	Result	Use This Instead	Compatibility Considerations
Naming the row times vector of a timetable when you specify the `'VariableNames'` name-value pair argument of the `timetable` function	Still runs	Not applicable	Starting in R2017b, the `timetable` function assigns the name of the input time vector as the name of the row times vector, even when you specify variable names using the `'VariableNames'` name-value pair argument. Previously in R2016b and R2017a, `timetable` assigned `'Time'` as the name of the row times vector when you specified `'VariableNames'`. This behavior changed because the row times vector is not a timetable variable. The row times are the labels of the rows. To rename the row times vector, use the `DimensionNames` property. For example, `TT.Properties.DimensionNames{1} = 'Time'` changes the name of the row times vector of `TT` to `'Time'`.
Subscripting into a timetable using datetime or duration values as row subscripts	Still runs	Not applicable	Starting in R2018a, timetable subscripting uses a tolerance to match subscripts to row times. Therefore, one subscript might match two or more distinct row times in a timetable when those row times differ only by a small amount. Previously in R2016b through R2017b, you could access data in a timetable using a datetime or duration array as the row subscript, but subscripting required an exact match between a time in the subscripts and a row time of the timetable. You can reproduce the earlier behavior that required an exact match in several ways. To select only exact matches at a single time, create a logical row subscript by testing for equality between subscripts and the row times using the equality operator, `==`. To select only exact matches at multiple times, create a logical row subscript using the `ismember` function. As an alternative, use the `withtol` function with a tolerance of `seconds(0)`.

App Building

App Designer: Create deployed web apps using MATLAB Compiler

If you have MATLAB Compiler™ installed on your system, you can create web apps using the **Share** button in the App Designer toolstrip. For more information, see Ways to Share Apps.

Google Chrome and Internet Explorer (version 11 and later) support running web apps.

App Designer: Add and configure tree components on the App Designer canvas

Drag and drop tree components from the **Component Library** onto the canvas, and change their text labels and positions directly in the canvas.

App Designer: Select from recently used argument sets when running apps with input arguments

Configure up to seven sets of input arguments under the **Run** button in the App Designer toolstrip. Select any of the sets to run them again. The sets remain in the drop-down list until you close the app.

App Designer: Edit axes title and label directly in the canvas

Now you can modify plot titles and axes labels more quickly by editing them directly on the canvas.

GUIDE: Migrate GUIDE apps to App Designer

For assistance in migrating your apps to App Designer, use the GUIDE to App Designer Migration Tool for MATLAB. This tool is available as a support package.

App Testing Framework: Author automated tests for App Designer apps

Use the app testing framework to write automated tests for your apps. The app testing framework leverages the MATLAB unit testing framework.

You can use the MATLAB app testing framework to test apps built with App Designer or apps built programmatically using the `uifigure` function. The app testing framework enables you to author a test class that programmatically performs a gesture on a UI component, such as pressing a button or dragging a slider, and verifies the behavior of the app.

For more information, see App Testing Framework.

Figure Objects: Maximize and minimize figures programmatically

Use the `WindowState` property to maximize, minimize, or display a figure in full-screen mode.

uitable Function: Specify data as table array

The `Data` property of `Table` UI components now accepts `table` arrays. For example:

```
T = readtable('patients.xls');
uit = uitable(uifigure,'Data',T);
```

`table` arrays are supported only in App Designer apps or figures created with the `uifigure` function.

uidatepicker Function: Add date selection controls to apps

Call the `uidatepicker` function to add a date picker to an app.

To display a date picker in an App Designer app, call the `uidatepicker` function from within a callback, such as the `StartupFcn` for the `UIFigure` component.

Date pickers work only in App Designer apps or in figures created with the `uifigure` function.

uiprogressdlg Function: Create modal in-app progress dialog boxes to apps

Call the `uiprogressdlg` function to create a progress dialog box within an app.

The dialog box can only display in App Designer apps or in figures created with the `uifigure` function.

uitree Function: Create trees with editable node text in the running app

Specify the `Editable` property of a tree so that users can change the node text while the app is running. Specify the `NodeTextChangedFcn` callback to make the app respond when the user changes node text.

Component Text Alignment: Improved text alignment for labels, check boxes, and radio buttons

The default height and vertical alignment for labels, check boxes, and radio buttons are now consistent with other single-line, text-based components. The new values make it easier to align the text of these components with other components.

Only labels, check boxes, and radio buttons in App Designer and in figures created with the `uifigure` function are affected.

Compatibility Considerations

The new default height for these components is 22 pixels (previously, it was 15 pixels). Text is now centered vertically within the component’s text box (previously, text was aligned to the top of the text box).

The labels, check boxes, and radio buttons in apps created in previous releases might look different when you open or run them in R2018a.You might need to change the vertical alignment, or adjust the height and location of those components to maintain a consistent appearance.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Apps saved in App Designer using R2018a	Error (when opening in previous releases)	Select **Save** > **Save Copy As** in the R2018a App Designer toolstrip.	The `.mlapp` file format changed in R2018a. Apps created or modified in App Designer return an error when you try to open them in previous releases. To modify an app across multiple releases, select **Save > Save Copy As** in the R2018a App Designer toolstrip. Then select the appropriate MATLAB release from the **Save as type** drop-down list. Alternatively, make a backup copy of the app before opening the app in R2018a.
Apps created in previous releases that contain labels, check boxes, or radio buttons	Still Runs	Not applicable	The new default height and vertical alignment of text for labels, check boxes, and radio buttons has changed. You might need to change the vertical alignment, or adjust the height and location of those components to maintain a consistent appearance. For more information, see “Component Text Alignment: Improved text alignment for labels, check boxes, and radio buttons” on page 5-30.

Performance

Startup: Increased speed of MATLAB startup time

MATLAB starts faster because of infrastructure improvements and optimizations.

Execution Engine: Execute tight loops with scalar math faster

Loops that contain mainly indexing and scalar math operations execute faster due to execution engine optimizations.

Execution Engine: Improved performance for common programming patterns

Common programming patterns, such as the evaluation of anonymous functions, and common combinations of operations, such as `isequal(size(x),size(y))`, execute faster due to targeted optimizations.

App Designer: Starting, loading, and layout tasks are faster

Starting App Designer is 40% faster than in R2017b. Loading apps and performing layout tasks in App Designer are 10% to 30% faster than in R2017b.

Hardware Support

Raspberry Pi: Support for Raspberry Pi Zero W board

You can now use the MATLAB Support Package for Raspberry Pi Hardware with the Raspberry Pi Zero W board.

MATLAB Support Package for Raspberry Pi Hardware enables you to communicate with Raspberry Pi Zero W using Wi-Fi® or a micro-USB cable connection. For more information on how to communicate with Raspberry Pi Zero W, follow the steps in Install, Update, or Uninstall Support Package (MATLAB Support Package for Raspberry Pi Hardware).

MATLAB Online: Acquire live images from USB webcams in MATLAB Online

You now can connect to and stream images from your USB webcam in MATLAB Online. Connect a USB webcam to your computer and give your browser access to your camera. All the functions in the MATLAB Support Package for USB Webcams are available for MATLAB Online.

Advanced Software Development

Tab Completion: Describe your function syntaxes for custom tab completion and other contextual suggestions

To customize code suggestions and completions for your functions, provide MATLAB with a JSON-formatted file that contains information about your function signatures. MATLAB uses this information to improve interactive features, such as tab completion and function hints. For more information, see Customize Code Suggestions and Completions.

Unit Testing Framework: Run tests from the MATLAB Editor toolstrip

You can run tests from the MATLAB Editor toolstrip. When you open a function-based or class-based test file, the Editor toolstrip has options to run all tests in the file or to run a single test in the file. Also, you can customize the test run with options, such as running tests in parallel (which requires Parallel Computing Toolbox) or running tests with a specified level of output detail.

The Run Tests controls in the toolstrip provide an alternative to programmatically running tests with the `runtests` function. For more information, see Run Tests in Editor.

App Testing Framework: Author automated tests for App Designer apps

Use the app testing framework to write automated tests for your apps. The app testing framework leverages the MATLAB unit testing framework.

You can use the MATLAB app testing framework to test apps built with App Designer or apps built programmatically using the `uifigure` function. The app testing framework enables you to author a test class that programmatically performs a gesture on a UI component, such as pressing a button or dragging a slider, and verifies the behavior of the app.

For more information, see App Testing Framework.

Unit Testing Framework: Rerun failed tests with one click

If a test failure is caused by incorrect or incomplete code for a test or for the code under test, it is useful to be able to rerun failed tests quickly and conveniently. If test failures exist in your test results, then MATLAB displays a link to rerun failed tests after it displays the number of failed tests.

```
Totals:
   1 Passed, 1 Failed (rerun), 0 Incomplete.
   0.25382 seconds testing time.
```

For more information, see Rerun Failed Tests.

Unit Testing Framework: Test if values point to existing files or folders with IsFile and IsFolder constraints

You can test if a value, specified as a string scalar or character vector, points to an existing file or folder.

- To test if a value points to an existing file, use `verifyThat`, `assertThat`, `assumeThat`, or `fatalAssertThat` with the `IsFile` constraint. For more information, see `matlab.unittest.constraints.IsFile`.
- To test if a value points to an existing folder, use `verifyThat`, `assertThat`, `assumeThat`, or `fatalAssertThat` with the `IsFolder` constraint. For more information, see `matlab.unittest.constraints.IsFolder`.

Unit Testing Framework: Test if two sets are the same with IsSameSetAs constraint

To test if two sets are the same, use the `IsSameSetAs` constraint. The testing framework considers tests to be the same if they contain the same elements. Sets can be the same even if they have different order, shape, and size. For example, if `S = {'a' 'b' 'c'}`, then set `S` is the same as the following sets.

```
S1 = {'a' 'b' 'c'};
S2 = {'c';'a';'b'};
S3 = {'a' 'b' 'c' 'c' 'b'};
```

For more information, see `matlab.unittest.constraints.IsSameSetAs`.

Unit Testing Framework: Select tests by test class hierarchy

You can run tests that have a specified superclass:

- To select and run test elements that have a specified superclass, use the `'Superclass'` name-value pair with the `runtests`, `runperf`, and `testsuite` functions or the `matlab.unittest.TestSuite` suite creation methods.
- To select test elements from an existing test suite, use the `TestSuite.selectIf` method with the `HasSuperclass` selector.

Compatibility Considerations

Test suite elements created prior to MATLAB R2018a cannot be filtered by test class hierarchy. To filter these test elements by superclass, recreate the test suite in R2018a or later.

Unit Testing Framework: Direct output stream to unique files for plugins

To direct text output from plugins to a unique file, use the `matlab.unittest.plugins.ToUniqueFile` output stream. This output stream is useful for running tests in parallel while redirecting output to a file, since it uses a unique file name for each instance of the output stream. The `ToUniqueFile` output stream is different from the `ToFile` output stream, which overwrites the file. For more information, see `matlab.unittest.plugins.ToUniqueFile`.

The `ToUniqueFile` output stream is supported for

- `matlab.unittest.plugins.DiagnosticsValidationPlugin`
- `matlab.unittest.plugins.FailureDiagnosticsPlugin`

- `matlab.unittest.plugins.TAPPlugin`
- `matlab.unittest.plugins.TestRunProgressPlugin`

Additionally, the `matlab.unittest.plugins.OutputStream` class provides an interface for test authors to create custom output streams.

Unit Testing Framework: Increased access to parameterized testing properties

When you create a parameterized test, you define properties in a `properties` block with a `TestParameter`, `MethodSetupParameter`, or `ClassSetupParameter` attribute, depending on the parameterization level for your test. You can now access “up-level” parameters.

- Tests in a `Test` method block can access parameters that you define in `TestParameter`, `MethodSetupParameter`, and `ClassSetupParameter` `properties` blocks.
- Tests in a `TestMethodSetup` method block can access parameters that you define in `MethodSetupParameter` and `ClassSetupParameter` `properties` blocks.
- Tests in a `TestClassSetup` method block can access parameters that you define in a `ClassSetupParameter` `properties` block.

In previous versions of MATLAB, you could access only parameters from tests in method blocks at the corresponding level. For example, parameters defined in a `MethodSetupParameter` property block were only accessible from tests in a `TestMethodSetup` method block. For more information, see Create Advanced Parameterized Test.

Unit Testing Framework: Compare cell arrays of character arrays using StringComparator

In addition to supporting strings and character arrays, the `StringComparator` constraint now supports cell arrays of character arrays. For more information, see `matlab.unittest.constraints.StringComparator`.

Compatibility Considerations

Update any tests that rely on the `StringComparator` constraint not being satisfied by equal cell arrays of character arrays. If the cell arrays of character arrays are equal, the comparator is satisfied. In previous versions of MATLAB, the comparator was not satisfied for any values that were cell arrays of character arrays. For example, the following test passes in MATLAB R2018a and later. In earlier versions of MATLAB, the test fails because the comparator did not support cell arrays of character arrays.

```
import matlab.unittest.TestCase
import matlab.unittest.constraints.StringComparator
import matlab.unittest.constraints.IsEqualTo

testCase = TestCase.forInteractiveUse;
actVal = {'coffee','cream','sugar'};
expVal = {'coffee','cream','sugar'};

testCase.verifyThat(actVal,IsEqualTo(expVal,'Using',StringComparator))
```

Unit Testing Framework: Comparison method for objects changed

As of R2018a, the `ObjectComparator` is satisfied if `isequaln` returns true. However, if the class of the expected value defines an `isequal` method, whether visible or hidden, but not an `isequaln` method, the `ObjectComparator` uses the `isequal` method for comparison. In previous releases, `ObjectComparator` used `isequal` to compare all objects unless the class of the expected value defined a visible `isequaln` method.

The `IsEqualTo` constraint and the `assertEqual`, `assumeEqual`, `fatalAssertEqual`, and `verifyEqual` qualification methods leverage `ObjectComparator` and, therefore, inherit the same change in behavior. For more information, see `matlab.unittest.constraints.ObjectComparator`.

Compatibility Considerations

The ObjectComparator now uses `isequaln` for comparison instead of `isequal` if the class of the expected value

- Defines `isequaln`, whether visible or hidden.
- Does not define either `isequal` or `isequaln`.

Performance Testing Framework: Define multiple, labeled measurement boundaries in test methods

Measurement boundaries enable you to refine which code the performance framework measures, and now you can label the measurement boundaries. Measurements from multiple boundaries with the same label in the same test method are accumulated and summed. The performance framework appends the label is to the test element name in the measurement results. For more information, see `startMeasuring`.

Mocking Framework: Specify default property values on mock object

When you create a mock object for a class, you can specify default values for properties that are implemented by the class. To specify default property values, use the `DefaultPropertyValues` name-value pair argument with the `createMock` method. For more information, see `matlab.mock.TestCase.createMock`.

Mocking Framework: Obtain interaction history for mock object

To obtain a history of recorded interactions for a mock object, use either the `matlab.mock.TestCase.getMockHistory` method or the `matlab.mock.InteractionHistory.forMock` method. When you call certain publicly visible methods or access or modify certain publicly visible properties on a mocked class, the mocking framework records the interaction. For more information, see `matlab.mock.TestCase.getMockHistory` or `matlab.mock.InteractionHistory.forMock`.

Mocking Framework: Construct mocks for classes that have Abstract properties with other attributes

You can construct a mock object for classes that have `Abstract` properties and other attributes. For example, you can construct a mock for a property that has Abstract and Constant attributes. The

mock implements the property as a concrete, `Constant` property. Similarly, properties with `Abstract` and `Hidden` attributes are implemented as concrete and `Hidden` properties. For more information, see Create Mock Object.

matlab.net.http Package: Stream data to and from a web service and handle forms and multipart messages

To act on or display streamed data while it is being received, use the `matlab.net.http.io.ContentConsumer` class. To obtain or generate the data at the same time it is being sent, use the `matlab.net.http.io.ContentProvider` class. Using this class avoids the need to have all the data in memory before the start of a message,.

To handle forms and multipart messages, use the `matlab.net.http.io.MultipartConsumer` and `matlab.net.http.io.MultipartProvider` classes.

The following classes and methods were added to the `matlab.net.http` and `matlab.net.http.field` packages.

- `matlab.net.http.HeaderField` methods — `addFields`, `changeFields`, `getFields`, `removeFields`, `replaceFields`
- `matlab.net.http.field.GenericField` methods — `getParameter`, `removeParameter`, `setParameter`
- `matlab.net.http.field.ContentDispositionField` — This class specifies a Content-Disposition header field, which is commonly used in multipart form requests.
- `matlab.net.http.field.GenericParameterizedField` — This class is a version of the `GenericField` class that supports parameterized syntax.

C++ MEX Interface: Access MATLAB data and objects easier from C++

Author MEX functions using modern C++ design patterns, extended data type support, and MATLAB copy-on-write semantics for faster handling of large data arrays. For more information, see C++ MEX Applications.

If you do not need MEX files that work in R2017b and earlier and you are familiar with modern C++, then consider using the new C++ MEX API and MATLAB Data API. If you are more comfortable working in the C language, continue using the C MEX API and C Matrix API.

Class Constructors: Author subclass without implementing a constructor solely to pass arguments through to a superclass constructor

MATLAB passes arguments implicitly from a default subclass constructor to the superclass constructor. This behavior eliminates the need to implement a constructor method for a subclass only to pass arguments to the superclass constructor. For more information, see Implicit Call to Inherited Constructor.

Property Validation: Get information about property validation

Get information on the validation defined for a property by accessing the validation metadata for that property. For more information, see Metadata Interface to Property Validation.

Property Validation: Define validation for abstract properties

As of MATLAB Version 9.4 (R2018a), you can define validation for abstract properties. The validation applies to all subclasses. For more information, see Abstract Property Validation.

Functions: Call numArgumentsFromSubscript for object dot method from overloaded subsref

In releases prior to MATLAB Version 9.4 (R2018a), the built-in `numArgumentsFromSubscript` function handled method calls of the form *`object.method`* incorrectly.

In releases prior to MATLAB Version 9.4 (R2018a), when a class overloaded `subsref` and `numArgumentsFromSubscript` and a user called an indexed expression of the form *`object.method()`*, MATLAB called `subsref` with `0` outputs without calling `numArgumentsFromSubscript`.

If instead the user called *`object.method`* (with no parentheses), then MATLAB treated this expression as a dot-reference and called `numArgumentsFromSubscript`. If an overloaded `numArgumentsFromSubscript` called the built-in version of the function, MATLAB returned `1` for the *`object.method`* expression. This caused errors when calling methods that return no outputs, and created differences in behavior between *`object.method()`* and *`object.method`*.

With MATLAB Version 9.4 (R2018a), when a method is called through a dot-indexing expression on the right-hand side, the built-in `numArgumentsFromSubscript` returns 0. Because of this change, MATLAB calls `subsref` with zero outputs in both the *`object.method()`* and *`object.method`* cases.

Compatibility Considerations

Changed behavior can occur for cases in which classes overload `numArgumentsFromSubscript` and the method calls the built-in function for indexing expressions that end in dot-method (*`object.method`*) references.

Classes: Concatenate matlab.lang.OnOffSwitchState enumeration members with nonmember char and string

With the release of MATLAB Version 9.4 (R2018a), the `matlab.lang.OnOffSwitchState` class supports special concatenation behavior to enable the formation of text expressions by concatenating enumeration members with character vectors or strings. For more information, see `matlab.lang.OnOffSwitchState`.

Compatibility Considerations

In releases prior to MATLAB Version 9.4 (R2018a), arrays containing `matlab.lang.OnOffSwitchState` enumeration members and character vector or string elements required that all array elements be convertible to enumeration members (that is, `on`, `off`, `true`, `false`, `1`, or `0`). The resulting array was of type `matlab.lang.OnOffSwitchState`. With the release of MATLAB Version 9.4 (R2018a), concatenation rules enable concatenation with character vectors or strings that do not map to enumeration members. For more information, see `Concatenation Rules for OnOffSwitchState`.

Python Version 3.4: Support discontinued

Support for Python version 3.4 is discontinued.

Compatibility Considerations

To ensure continued support for your applications, upgrade to a supported version of Python—version 3.5 or 3.6.

Source Control Integration: View changes, save revisions, and manage repository locks

When you are using SVN source control, the SVN file revisions history now shows a tree view of change set files that has a context menu. You can select a particular file revision and view changes or save revisions. For more information, see Review Changes in Source Control.

You can now monitor and break SVN locks. The **SVN Repository Locks** dialog box supports SVN locking workflows within teams:

- Determine who has a lock
- Break locks
- Group locks by user or file

For details, see Get SVN File Locks.

MATLAB Engine API for C++: Set and get a property value on an object in an object array

With release R2018a, C++ Engine applications can call the `matlab::engine::MATLABEngine` `getProperty`, `getPropertyAsync`, `setProperty`, and `setPropertyAsync` member functions with object array inputs and pass an array index to access scalar objects within the array. In previous releases, C++ Engine applications could access object properties only from scalar object variables. For more information, see Pass Variables from MATLAB to C++.

MATLAB Data API: Applications built with R2018a API do not run in MATLAB R2017b

Applications using the MATLAB Data API built in R2018a are not supported in MATLAB R2017b.

Compatibility Considerations

Applications built in MATLAB R2018a run only in MATLAB R2018a. For more information, see Version Compatibility.

MEX Functions: Build C MEX Files with Interleaved Complex API

As of MATLAB Version 9.4 (R2018a), MATLAB uses an interleaved storage representation of complex numbers. The term Interleaved complex refers to this representation, where the real and imaginary

parts are stored together. For more information, see MATLAB Support for Interleaved Complex API in C MEX Functions.

This change does not affect the MATLAB language. You can continue to use the functionality described in Complex Numbers without any modification of your functions and scripts.

Compatibility Considerations

If you build C MEX functions, C/C++ MEX S-functions, or standalone MATLAB engine and MAT-file applications, then you should review the Do I Need to Upgrade My MEX Files to Use Interleaved Complex API? topic. MATLAB does not support the interleaved complex API for Fortran functions.

The functionality for `mxGetPr (C)`, `mxSetPr (C)`, `mxGetPi (C)`, `mxSetPi (C)`, `mxGetData (C)`, `mxSetData (C)`, `mxGetImagData (C)`, `mxSetImagData (C)`, and `mxGetElementSize (C)` has changed. For more information, see “Functionality being removed or changed” on page 5-43.

MEX Functions: Release-specific build options

The `mex` command has new build options, `-R2017b` and `-R2018a`, which link with release-specific versions of the C Matrix API.

- `-R2017b` — Default option. This option is equivalent to the command:

  ```
  mex mymex.c -largeArrayDims -DMEX_DOUBLE_HANDLE
  ```

- `-R2018a` — Uses the interleaved complex API, which includes the typed data access functions. For more information, see MATLAB Support for Interleaved Complex API in C MEX Functions.

Version Embedded in MEX Files

The `mex` command embeds a MEX version number in MEX files built by MATLAB R2016b and later. This number identifies the version of the Matrix API that the MEX function expects to link against at runtime.

Compatibility Considerations

If you do not use the `mex` command to build your MEX files, then you must update the commands you use to build MEX files. For more information, see https://www.mathworks.com/matlabcentral/answers/377799-compiling-mex-files-without-the-mex-command.

Perl 5.26.1: MATLAB support

MATLAB ships with Perl version 5.26.1.

- See www.perl.org for a standard distribution of perl, perl source, and information about using perl.
- See https://metacpan.org/pod/HTML::Parser for a standard distribution of HTML::Parser, source code, and information about using HTML::Parser.
- See https://metacpan.org/pod/HTML::Tagset for a standard distribution of HTML:Tagset, source code, and information about using HTML:Tagset.

Compatibility Considerations

If you use the `perl` command on Windows platforms, see www.perl.org for information about using this version of the Perl programming language.

System objects: Create System Objects in MATLAB

System objects are a specialized kind of MATLAB object that allow you to easily implement and simulate dynamic systems. You can use predefined System objects shipped with many System Toolboxes. You can also create your own System objects in the MATLAB editor. For more information, see Define Basic System Objects.

When you create System objects in R2018a, by default, users of that System object can change characteristics of inputs, discrete states, tunable properties from call to call. New and updated methods allow you to restrict these characteristics:

Characteristic That Can Change	Methods to Restrict Characteristic
number of inputs	`getNumInputsImpl`
number of outputs	`getNumOutputsImpl`
input size	`isInputSizeMutableImpl`
input data-type	`isInputDataTypeMutableImpl`
input complexity	`isInputComplexityMutableImpl`
discrete state data-types	`isDiscreteStateSpecificationMutableImpl`
tunable property data-types	`isTunablePropertyDataTypeMutableImpl`

For more information, see System Objects.

Compatibility Considerations

If you want to retain strict rules for inputs, tunable properties, and discrete states, use the `sysobjupdate` function to update code in existing System objects. For more information, type `help sysobjupdate` at the MATLAB command line.

The `isInputSizeLocked` method will be removed in a future release. Use `isInputSizeMutableImpl` instead.

System object support for strings

System objects accept strings as inputs for text input and property values.

When authoring a System object, you can use strings to define a StringSet property. However, the default value of a StringSet property must be defined as a character vector.

.NET: Supports string data type

When calling a .NET method or function, MATLAB converts `string` scalar arguments to a .NET `System.String` object and `string` array arguments to `System.String[]`. For more information, see Pass Data to .NET Objects.

The MATLAB `string` function converts `System.String` scalar arguments to a `string` scalar. The function converts `String.String[]`, `String.String[,]`, and so on, to MATLAB `string` arrays with the same dimensions and sizes. Conversion of jagged arrays, for example `String.String[][]`, is not supported. For more information, see Handle Data Returned from .NET Objects.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	GNU® gcc and gfortran version 6.x. Version 6.3 recommended.	Linux
Discontinued	GNU gcc and gfortran version 4.9.x	Linux
Discontinued	Apple Xcode 7.x	macOS
Discontinued	Intel C++ Composer XE 2013	Windows
Discontinued	Intel Visual Fortran Composer XE 2013	Windows
Discontinued	Intel Fortran Composer XE 2013	macOS
To be phased out	Visual C++ 2013 Professional	Windows

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Programmatic dependence on specific diagnostic subclass from `getDiagnosticFor` method of constraint and tolerances	Warns	Not applicable	Rework code that relies on properties or methods specific to `matlab.unittest.diagnostic.ConstraintDiagnostic` instances returned from the `getDiagnosticFor` method of `matlab.unittest.constraints` classes. In a future release, diagnostics returned from constraint and tolerance classes in the `matlab.unittest.constraints` package will be subclasses of `matlab.unittest.diagnostics.Diagnostic` and might not be instances of `matlab.unittest.diagnostic.ConstraintDiagnostic`.
Negation of `matlab.unittest.constraints.ReturnsTrue` constraint (`~ReturnsTrue`)	Errors	Not applicable	Change the logic for tests that rely on negating the `ReturnsTrue` constraint.

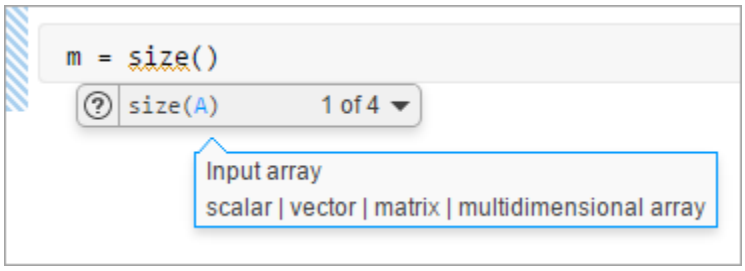
Functionality	Result	Use This Instead	Compatibility Considerations
`mxGetPi` and `mxSetPi` in C MEX functions built with the interleaved complex API (`mex` option `-R2018a`)	Errors	Use `mxGetComplexDoubles` instead of `mxGetPr` and `mxGetPi`.	MATLAB Support for Interleaved Complex API in C MEX Functions
`mxGetImagData` and `mxSetImagData` in C MEX functions built with the interleaved complex API (`mex` option `-R2018a`)	Errors	Typed Data Access functions	MATLAB Support for Interleaved Complex API in C MEX Functions
`mxGetPr` and `mxSetPr` in C MEX functions built with the interleaved complex API (`mex` option `-R2018a`)	Still runs when called for real `mxArrays`. Errors when called for complex `mxArrays`.	Typed Data Access functions	MATLAB Support for Interleaved Complex API in C MEX Functions
`mxGetData` and `mxSetData` in C MEX functions built with the interleaved complex API (`mex` option `-R2018a`)	Still runs	Typed Data Access functions	Use `mxGetData (C)` and `mxSetData (C)` for nonnumeric arrays only.
`mxGetElementSize` in C MEX functions built with the interleaved complex API (`mex` option `-R2018a`)	Still runs	Not applicable	`mxGetElementSize (C)`
`mexSetTrapFlag` in C and Fortran Matrix API	Errors	`mexCallMATLABWithTrap`	`mexCallMATLABWithTrap` lets you catch, or trap, errors.

R2017b

Desktop

Live Editor: Write MATLAB commands with automated, contextual hints for arguments, property values, and alternative syntaxes

When writing commands, MATLAB automatically displays contextual hints for arguments, property values, and alternative syntaxes. For example, if you want to use the `size` function, MATLAB displays the syntax information to help you write the command.



MATLAB also automatically suggests and completes the names of functions, models, MATLAB objects, files, folders, variables, structures, graphics properties, parameters, and options. For more information, see Automatic Code Suggestions and Completions.

Live Editor: Export live scripts to LaTeX format

To export your live script to LaTeX format, on the **Live Editor** tab, select **Save** > **Export to LaTeX**. MATLAB creates a separate Extensible Stylesheet Language (XSL) file in the same folder as the output document, if one does not exist already. XSL files give you more control over the appearance of the output document. For more details, see Share Live Scripts.

Live Editor: Display high-resolution plots in PDF output

When saving a live script as a PDF, MATLAB now includes high-resolution plots in the PDF file.

Live Editor: Horizontally align text, equations, and images

You can horizontally align text, equations, and images in live scripts. On the **Live Editor** tab, in the **Text** section, select left, center, or right.

For more information, see Format Live Scripts.

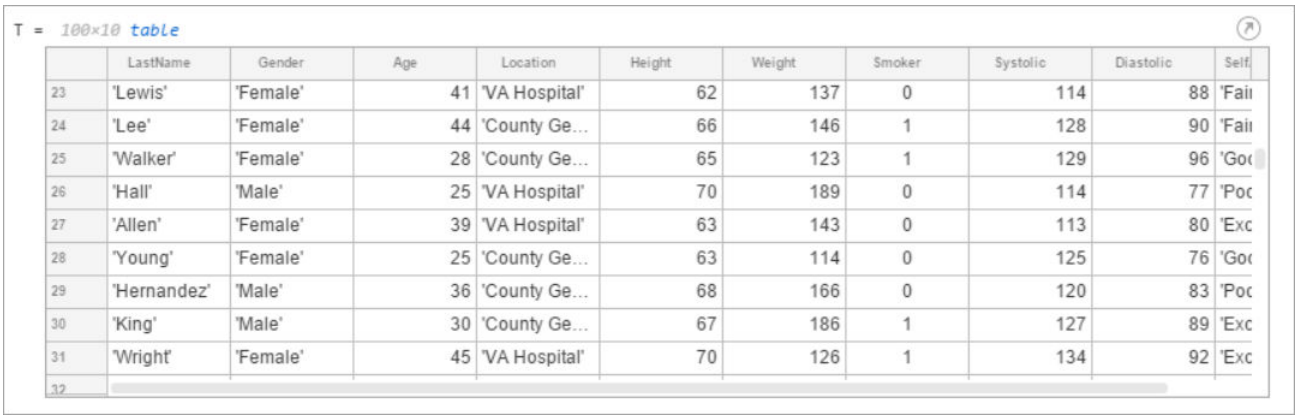
Live Editor: Automatically match delimiters and wrap comments while editing code

MATLAB autocompletes parentheses and quotes when entering code in the Live Editor. For example, if you type `r = rand(`, MATLAB automatically adds the closing parentheses to the statement (`r = rand()`). The Live Editor also autocompletes any comments, character vectors, and strings that are split across two lines.

For more information, see Run Code.

Live Editor: View and scroll through table data, including variable and row names

Use scroll bars to explore data in tables and timetables in live scripts. View the variable and row names included in the tables.


T = 100×10 table

	LastName	Gender	Age	Location	Height	Weight	Smoker	Systolic	Diastolic	Self
23	'Lewis'	'Female'	41	'VA Hospital'	62	137	0	114	88	'Fai
24	'Lee'	'Female'	44	'County Ge...	66	146	1	128	90	'Fai
25	'Walker'	'Female'	28	'County Ge...	65	123	1	129	96	'Goo
26	'Hall'	'Male'	25	'VA Hospital'	70	189	0	114	77	'Poo
27	'Allen'	'Female'	39	'VA Hospital'	63	143	0	113	80	'Exc
28	'Young'	'Female'	25	'County Ge...	63	114	0	125	76	'Goo
29	'Hernandez'	'Male'	36	'County Ge...	68	166	0	120	83	'Poo
30	'King'	'Male'	30	'County Ge...	67	186	1	127	89	'Exc
31	'Wright'	'Female'	45	'VA Hospital'	70	126	1	134	92	'Exc
32										

Live Editor: Check code for errors and warnings using the message bar and message indicator

Determine whether a live script contains an error or warning using the message indicator. Navigate through errors and warnings using the message bar. For more information, see Check Code for Errors and Warnings.

Documentation: Use the Live Editor in a web browser to open, edit, and run MATLAB online documentation examples

Open, edit and run live script examples directly from the MATLAB online documentation using the Live Editor in a web browser. To open an example, click the **Try this Example** button to the right of the example and select **Try it in your browser**.

MATLAB Drive: Store, access, and manage your files from anywhere

If you have MATLAB Drive Connector installed on your system, you can access files and folders in your drive from MATLAB using the Current Folder browser.

To view your recent activity, in the Current Folder browser, right-click any MATLAB Drive file or folder and select **MATLAB Drive > View Recent Activity...**. To open MATLAB Drive online, right-click any MATLAB Drive file or folder and select **MATLAB Drive > Go to MATLAB Drive Online...**.

For more information, see Manage Files and Folders or the MATLAB Drive documentation.

Add-On Manager: Customize your MATLAB environment by enabling and disabling add-ons

Enable and disable add-ons in the Add-On Manager to customize your MATLAB environment. To enable an add-on in the Add-On Manager, click the ⋮ button to the right of the add-on and select **Enabled**. This option is only available for apps, toolboxes, functions, collections and Simulink models.

You also can enable or disable the add-on using the `matlab.addons.enableAddon` and `matlab.addons.disableAddon` functions.

Add-On Manager: Find installed add-ons faster using sort and search

Use sort and search in the Add-On Manager to find installed add-ons more efficiently.

Toolbox Packaging: Create a Getting Started Guide for your toolbox from a Live Script template

You can create a Getting Started Guide when you package your toolbox. Users of your toolbox can view the Getting Started Guide through the Options menu for the toolbox in the Add-On Manager. After you select your toolbox folder, the option to create a Getting Started Guide appears in the **Examples, Apps, and Documentation** section of the Package a Toolbox dialog box. For more information, see Create and Share Toolboxes.

Toolbox Packaging: Share your toolbox on File Exchange directly when you package it

When you are creating your toolbox, you can package the toolbox and share it on MATLAB Central File Exchange. Select **Package and Share** from the **Package** menu at the top of the Package a Toolbox dialog box. This option opens a web page for your toolbox submission to File Exchange submission. MATLAB populates the File Exchange submission form with information from the Package a Toolbox dialog box. Review and submit the form to share your toolbox on File Exchange. For more information, see Create and Share Toolboxes.

Command Window: View updated display for cell arrays

When displaying cell array outputs, the Command Window now displays curly braces around each cell. The curly braces show the delimiters of the cell and help unify the display of the cell array.

Use curly braces to access the contents of a cell. For example, this code creates a 2-by-3 cell array of text and numeric data, and then displays the contents of the last cell in the cell array.

```
C = {'one', 'two', 'three'; 1, 2, 3}
last = C{2,3}
```

```
C =
  2×3 cell array
    {'one'}    {'two'}    {'three'}
    {[  1]}    {[  2]}    {[    3]}

last =
     3
```

Additionally, if a cell contains a numeric empty value, then the Command Window displays the class and size of the cell.

For more information, see Create Cell Array.

Language and Programming

Code Compatibility Report: Generate a report that helps the updating of code to a newer MATLAB release

You can generate a report of potential compatibility issues in your code using the `codeCompatibilityReport` function. For example, the report contains information about the use of discouraged or removed functions in your code and the occurrence of invalid syntaxes. After you upgrade to a newer version of MATLAB, you can use this report to identify potential compatibility issues in your existing code.

Alternatively, you can create a `CodeCompatibilityAnalysis` object to save results using the `analyzeCodeCompatibility` function.

isStringScalar Function: Determine whether input is a string array with one element

To determine whether the input argument is a string array that has only one element, use the `isStringScalar` function.

convertStringsToChars and convertCharsToStrings Functions: Enable your code to accept all text types as inputs without otherwise altering your code

To make your existing code accept strings as input arguments, you can use the `convertStringsToChars` function on the entire input argument list. `convertStringsToChars` converts input string arrays to character vectors or cell arrays of character vectors while returning all the other input arguments unaltered. If you add `convertStringsToChars` to the beginning of your code, then you do not need to make any other changes to accept strings as inputs.

Similarly, `convertCharsToStrings` converts input character vectors or cell arrays of character vectors to string arrays while returning the other input arguments unaltered. If you have code that works with strings, then you can add `convertCharsToStrings` to the beginning so that it also accepts character arrays as inputs.

arrayfun, cellfun, and structfun Functions: Return object arrays as output arguments

The `arrayfun`, `cellfun`, and `structfun` functions can return object arrays of any data type, so long as the objects can be concatenated.

Scripts: Run sections in scripts containing local functions

You can run an individual section in a script that contains local functions. To run a section, on the **Editor** or **Live Editor** tab, click  **Run Section**.

Note You must fix all syntax errors in the script before running an individual section.

isfile and isfolder Functions: Determine if input is a file or a folder

Use the `isfile` and `isfolder` functions to determine if an input is a file or a folder located on the specified path or in the current folder.

For example, this code creates a folder, and then it uses the `isfile` and `isfolder` functions to check whether the input is a file or folder:

```
mkdir myfolder;
result1 = isfile('myfolder')
result2 = isfolder('myfolder')
```

```
result1 =
  logical
   0

result2 =
  logical
   1
```

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
The `cellstr` function converts missing values in an input string array to 0-by-0 character arrays in the output cell array.	Still runs	Not Applicable	Starting in R2017b, `cellstr` converts missing values in string arrays to 0-by-0 character arrays. In R2016b and R2017a, `cellstr` throws an error if the input argument is a string array that contains missing values.
`isdir`	Still runs	`isfolder`	Replace all instances of `isdir` with `isfolder`. `isdir` searches for folders on the search path, which can lead to unexpected results. `isfolder` searches for folders only on the specified path or in the current folder.
On Windows, `tempdir` checks first for the existence of `TMP`, and then for `TEMP`.	Still runs	Not Applicable	On Windows, the `tempdir` function now checks first for the existence of the `TMP` environment variable, and then it checks for the `TEMP` environment variable. It uses the first path found. Previously, `tempdir` checked first for the existence of `TEMP`.
On Linux and Mac, `tempdir` function uses `TMPDIR`.	Still runs	Not Applicable	On Linux and Mac, the `tempdir` function now uses the `TMPDIR` environment variable, if it is defined.

Functionality	Result	Use This Instead	Compatibility Considerations

Using a MATLAB identifier to call a function, and then using it as a variable name inside the same function	Warns	Choose different MATLAB identifiers for local or imported functions and variable names.	In a future release, using the same identifier for a local or imported function, and then using it for a variable name will result in an error. For example, in a future release, both of the following functions will result in an error.
			`function myFunc()`
			`    x = containers.Map;`
			`    containers = 1;`
			`end`
			`function myFunc2()`
			`    import java.lang.*`
			`    % includes java.lang.String`
			`    s = String('Hello');`
			`    String = 1;`
			`end`

Functionality	Result	Use This Instead	Compatibility Considerations

Indexing into an implicitly defined variable in a function	Still runs	Explicitly define a variable before indexing in to it.	Currently, an identifier without an explicit declaration is classified as a variable when it is indexed with the colon, `end`, or curly braces. For example, `X(a,b,:)`, `X(end)`, or `X{a}`. In a future release, if a function of the same name exists on the path, the variable will be classified as a function.
			
			Consider the following code:
			
			`function myfunc()`
			`  load data.mat;`
			`  disp(x(:))`
			`end`
			
			If you intend to use `x` as a variable from `data.mat` instead of a function, explicitly define it.
			
			`function myfunc()`
			`  load data.mat x;`
			`  disp(x(:))`
			`end`
			
			Similarly, to use `x` as a variable obtained from a script, define the variable before invoking the script. This behavior also applies if the variable is implicitly introduced by any of the following functions: `sim`, `eval`, `evalc`, and `assignin`. For example, if `myscript.m` defines `x`, define `x` before calling `myscript`.
			
			`function myfunc2()`
			`  x = [];`
			`  myscript;`
			`  disp(x(:))`
			`end`

Functionality	Result	Use This Instead	Compatibility Considerations

Sharing uninitialized variables between a nested function and the parent function	Still runs	Explicitly define shared variables in the parent function before calling the nested function.	Consider the following code.
			
			`function myfunc()`
			`    nested();`
			`    x;`
			`    function nested()`
			`        x = 1;`
			`    end`
			`end`
			
			Currently, if `x` is a function on the path, MATLAB classifies it as a function in `myfunc` and a variable in `nested`. Otherwise, MATLAB classifies it as a variable shared between `nested` and `myfunc`.
			
			To avoid potential function behavior differences depending on the state of your MATLAB path, explicitly define shared variables in the parent function.
			
			`function myfunc()`
			`    x = 0;`
			`    nested();`
			`    x;`
			`    function nested()`
			`        x = 1;`
			`    end`
			`end`

Functionality	Result	Use This Instead	Compatibility Considerations

Nested functions in MATLAB do not inherit import statements from the parent function.	Still runs	Not Applicable	The `plot` statement in the following `nested` function calls the `plot` function on the path, not the `pkg.plot` function.
			
			In the future, MATLAB will call the `plot` function in `pkg.plot`.
			
			`function myfunc`
			`import pkg.plot`
			`  function nested`
			`  plot`
			`  end`
			`end`
			
			Similarly, in the following example, even if the `plot` function is defined in `pkg.*`, MATLAB calls the function on the path. However, in the future, MATLAB will call the function in imported package `pkg.*`.
			
			`function myfunc`
			`import pkg.*`
			`  function nested`
			`  plot`
			`  end`
			`end`

`ans` in nested functions	Still runs	Not Applicable	Nested functions now access and modify the `ans` variable that is defined in the parent function. In future releases, nested functions will have access to the `ans` variable even if the variable is declared implicitly by not suppressing the output. For example in future releases, in this code, `ans` returns 1.
			
			`function foo`
			`  a=1;`
			`  a  % Assigns “ans” to 1.`
			`  nested()`
			`  function nested`
			`  ans % will be 1`
			`  end`
			`end`

Using `ans` as a function name	Warns	Not Applicable	Rename all functions with the name `ans`. In a future release, using `ans` as a function name issues an error.
Declaring a variable as global after referencing it	Warns	Declare variables as global before using them.	Currently, if you declare a variable as global after referencing it, MATLAB issues a warning and the variable becomes a global variable. In a future release, this will result in an error.

Functionality	Result	Use This Instead	Compatibility Considerations
MATLAB disables folder change detection for all of *`matlabroot`*	Still runs	Not applicable	MATLAB now disables folder change detection for all of the files and folders in the *`matlabroot`* folder. Previously, MATLAB only disabled folder change detection for the files and folders in *`matlabroot`*`/Toolbox`. For more information, see Toolbox Path Caching in MATLAB.

Mathematics

decomposition Object: Solve linear systems repeatedly with improved performance

`decomposition` objects efficiently store a matrix decomposition for repeated use in solving linear systems $Ax = b$ or $xA = b$. These objects enable you to leverage the performance benefits of precomputing the matrix decomposition, but they *do not* require knowledge of how to use the matrix factors.

lsqminnorm Function: Find minimum-norm solution of underdetermined linear system

For underdetermined least-squares problems `A*x ≈ b` where several equivalent least-squares solutions exist, `x = lsqminnorm(A,b)` returns the least-squares solution `x` that minimizes `norm(x)` among all vectors that minimize `norm(A*x-b)`.

Previously, this functionality was available in a limited fashion using `x = pinv(A)*b`. However, `lsqminnorm` is faster since it does not need to compute the pseudoinverse `pinv(A)`, and it also supports sparse matrices.

dissect Function: Reorder sparse matrix columns using nested dissection ordering

Use the `dissect` function to generate fill reducing orderings of sparse matrices. The fill reducing ordering produced by `dissect` tends to reduce storage and computation time for sparse matrix factorizations when compared to other reordering functions such as `amd`.

vecnorm Function: Compute vector-wise norms of arrays

Use `vecnorm` to compute the norm of the rows and columns of a numeric N-dimensional array. Unlike the `norm` function that computes the matrix norm, `vecnorm` treats the elements along a specified dimension as vectors and calculates the norm of each vector.

polyshape Object: Create, analyze, and visualize 2-D polygons

The `polyshape` function enables you to create a polygon from a list of 2-D vertices. For example, `p = polyshape([0 0 1 1],[0 1 1 0])` creates a square with vertices (0,0), (0,1), (1,1), and (1,0).

After you create a `polyshape` object `p`, use the command `plot(p)` to visualize it. You also can compute common geometric quantities of `p`, such as its centroid and area. You can transform `p` using translation, rotation, or scaling, and you can query properties of `p` such as its number of boundaries or holes. If you are working with multiple polygons, then you can compute quantities, such as their intersection and union.

eigs Function: Improved algorithm and new options

The `eigs` function has a new algorithm, more intuitive behavior, and new options to simplify usage.

- **New `sigma` options**

  This table summarizes the new, more descriptive options for `sigma`. These changes do not require any updates to existing code since the old values continue to be supported.

Old `sigma`	New `sigma`	Which eigenvalues?
`'LM'`	`'largestabs'`	`max(abs(d))`
`'SM'`	`'smallestabs'`	`min(abs(d))`
`'LA'` (if real symmetric), `'LR'` (otherwise)	`'largestreal'`	`max(real(d))`
`'SA'` (if real symmetric), `'SR'` (otherwise)	`'smallestreal'`	`min(real(d))`
`'BE'` (if real symmetric)	`'bothendsreal'`	Combination of values returned by `'largestreal'` and `'smallestreal'`
`'LI'` (if complex)	`'largestimag'`	`max(imag(d))`
`'SI'` (if complex)	`'smallestimag'`	`min(imag(d))`
`'LI'` (if real nonsymmetric)	`'bothendsimag'`	Combination of values returned by `'largestimag'` and `'smallestimag'`

- **`sigma` values `'LR'`, `'LA'`, `'SR'`, `'SA'`, and `'BE'` accept both symmetric and nonsymmetric matrices**

  If an eigenproblem is symmetric, it has real eigenvalues. For that reason `'LR'`/`'LA'` and `'SR'`/`'SA'` now are considered to be equivalent by `eigs`.

  In the nonsymmetric case, the `'BE'` option uses the real part of each eigenvalue to determine which eigenvalues to return. The behavior is unchanged for problems that previously allowed the `'BE'` option.

- **Alternative name-value pairs for options**

  This table contains a one-to-one mapping of the old option structure fields to the corresponding new name-value pairs. These changes do not require any updates to existing code since the options structure continues to be supported.

Options Structure Field	Name-Value Pair	New Behaviors
`issym`	`'IsFunctionSymmetric'`	This option is used only for function handle inputs. For matrix inputs, `eigs` ignores this option.
`isreal`	-	`eigs` ignores this option because the new algorithm does not need to know whether the input matrix is real or complex.
`tol`	`'Tolerance'`	New default value of `1e-14`.
`maxit`	`'MaxIterations'`	

Options Structure Field	Name-Value Pair	New Behaviors
p	'SubspaceDimension'	
v0	'StartVector'	eigs uses the same starting vector for each call so that the output is reproducible between calls.
disp	'Display'	A display value of 2 no longer returns timing information. Instead, eigs treats a value of 2 the same as a value of 1. Also, the messages shown by the 'Display' option have changed. The new messages show the residual in each iteration, instead of the Ritz values.
spdB	'IsSymmetricDefinite'	New option.
cholB	'IsCholesky'	
permB	'CholeskyPermutation'	
fail	'FailureTreatment'	New option.

Compatibility Considerations

The eigs function also has some new behaviors that affect the output and might require updates to code.

- **Changes to sorting order of output**

  eigs now sorts the output according to the value of sigma. For example, the command eigs(A,k,'largestabs') produces k eigenvalues sorted in descending order by magnitude.

  Previously, the sorting order of the output produced by eigs was not guaranteed.

- **Reproducibility**

  Calling eigs multiple times in succession now produces the same result. Set 'StartVector' to a random vector to change this behavior.

- **Display**

  A display value of 2 no longer returns timing information. Instead, eigs treats a value of 2 the same as a value of 1. Also, the messages shown by the 'Display' option have changed. The new messages show the residual in each iteration, instead of the Ritz values.

svds Function: Set options with name-value pairs

You can use name-value pairs to set options with svds instead of using an options structure. This table contains a one-to-one mapping of the old option structure fields to the corresponding new name-value pairs.

These changes do not require any updates to existing code since the options structure continues to be supported.

Options Structure Field	Name-Value Pair
`tol`	`'Tolerance'`
`maxit`	`'MaxIterations'`
`p`	`'SubspaceDimension'`
`u0`	`'LeftStartVector'`
`v0`	`'RightStartVector'`
`fail`	`'FailureTreatment'`
`disp`	`'Display'`

Interpolation Functions: Method for modified Akima cubic Hermite interpolation

The `griddedInterpolant`, `interp1`, `interp2`, `interp3`, and `interpn` functions now support the interpolation method `'makima'`. This modified Akima cubic Hermite interpolation method has these properties:

- It is C^1 continuous.
- It produces fewer undulations than `'spline'`, but the result is not as aggressively flattened as `'pchip'`.
- Unlike `'pchip'`, it supports N-D arrays.
- Unlike `'spline'`, it does not produce overshoots.

convn Function: Compute convolutions on multidimensional arrays with improved performance

You can compute convolutions on multidimensional arrays faster with the `convn` function.

subgraph and highlight Functions: Specify graph nodes with logical vector

The `subgraph` and `highlight` functions for network analysis now accept a logical vector to select nodes.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`[bins,edges] = discretize(X,dur)` where `X` is a datetime or duration array	Still runs	N/A	In R2017a and later releases, `discretize(X,dur)` operating on a datetime or duration array `X` uses a maximum of $2^{16} = 65536$ bins. This aligns the behavior of `discretize` with that of `histcounts` and `histogram`. In releases before R2017a, `discretize` did not impose this limitation on the number of bins in the result.

Graphics

geobubble Function: Create an interactive map with bubbles whose size and color vary with data values

Create an interactive map with bubbles whose size and color vary with data values using `geobubble`.

wordcloud Function: Display words at different sizes based on frequency or custom size data

Display words at different sizes based on frequency or custom size data using `wordcloud`.

binscatter Function: Visualize data density with dynamic bin size adjustment

Use `binscatter` to visualize binned scatter plots and identify trends and relationships in arrays of data. `binscatter` produces a plot that is similar to a 2-D histogram (`histogram2`). However, since the `binscatter` function has custom zoom behavior that increases the resolution as you zoom in, it is better suited for visual exploration of data.

Tall Array Support: Visualize out-of-memory data using plot, scatter, and binscatter

The `plot`, `scatter`, and `binscatter` functions now support tall arrays. These functions plot the data in iterations, progressively adding to the plot as more data is read. Zooming and panning is supported during the updating process before the plot is complete.

heatmap Function: Sort rows and columns and use custom labels in a heatmap

You can make new types of modifications to a heatmap, such as reordering or relabeling the values along each axis. Use the new `HeatmapChart` listed in this table. If you want to reorder the values, you also can use the `sortx` and `sorty` functions.

Property	Description
`XData` and `YData`	Values associated with the rows or columns in `ColorData`.
`XDisplayData` and `YDisplayData`	Order of values along the *x*-axis or *y*-axis. Use these properties to view, rearrange, or show only a subset of the values.
`XDisplayLabels` and `YDisplayLabels`	Labels for *x*-axis or *y*-axis values. Use these properties to view or relabel the values.
`XLimits` and `YLimits`	First and last value displayed along the *x*-axis or *y*-axis.

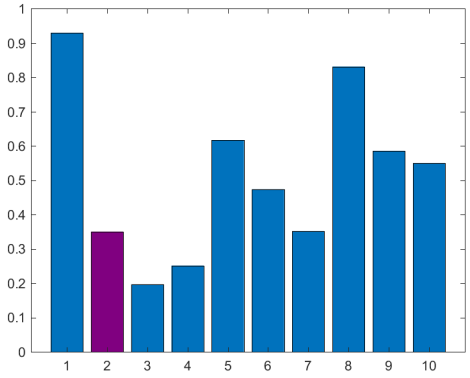
Property	Description
`ColorDisplayData`	Values as they appear in the heatmap, sorted based on `XDisplayData` and `YDisplayData`. This is a read-only property.

For an example, see Create Heatmap from Tabular Data.

bar Function: Control individual bar colors

Specify a different color for each bar using the new `CData` property for `Bar` objects. For example, create a bar chart and change the color of the second bar.

```
b = bar(rand(10,1));
b.FaceColor = 'flat';
b.CData(2,:) = [.5 0 .5];
```

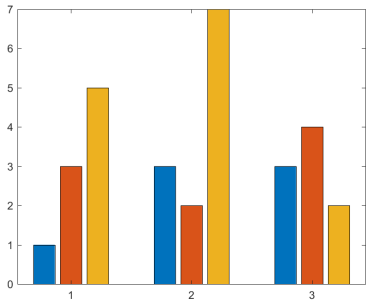
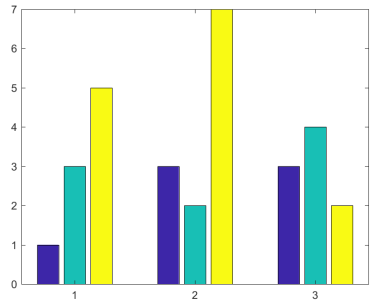


For more information, see `Bar Properties`.

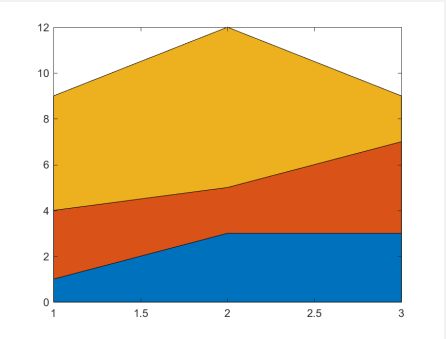
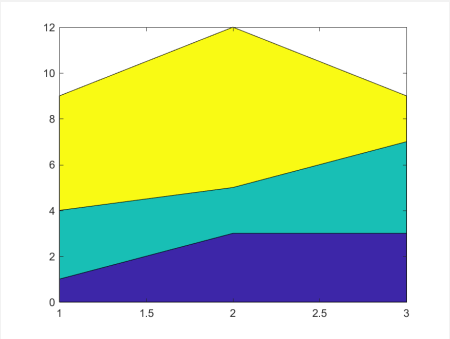
Chart Colors: Create bar and area charts with new default colors

Similar to line charts, bar charts and area charts now use colors from the `ColorOrder` property of the `Axes` object. Previously, the charts used colors from the colormap.

This table shows the color difference for bar charts.

R2017b and Later (ColorOrder)	Previous Releases (Colormap)
	

This table shows the color difference for area charts.

R2017b and Later (ColorOrder)	Previous Releases (Colormap)
	

Compatibility Considerations

Bar charts and area charts use different colors than in previous releases.

- To create a bar chart that uses colormap colors, set the `FaceColor` property to `'flat'`. Then set the `CData` property for each `Bar` object to an integer. For example:

  ```
  y = [1 3 5; 3 2 7; 3 4 2];
  b = bar(y,'FaceColor','flat');
  for k = 1:size(y,2)
      b(k).CData = k;
  end
  ```

- To create an area chart that uses colormap colors, set the `FaceColor` property to `'flat'`. For example:

  ```
  y = [1 3 5; 3 2 7; 3 4 2];
  area(y,'FaceColor','flat')
  ```

Axes Object: Specify the target axes for more functions

These functions now support an `Axes` object as the first input argument: `camlight`, `hidden`, `ishold`, `lightangle`, `lighting`, and `material`.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Default colors for bar and area charts	Still runs	Not applicable	Bar and area charts have new default colors. These charts use colors from the color order instead of colormap colors. For more information, see “Chart Colors: Create bar and area charts with new default colors” on page 6-19.
Default axis limits for bar charts, histograms, and graph plots	Still runs	Not applicable	Bar charts, histograms, and graph plots might use slightly different limits than in previous releases. These charts automatically choose limits that center the data. If you want to specify different limits, use the `xlim` and `ylim` functions.
`barh` function with the `Horizontal` parameter	Warns	`bar` function	Remove instances of code that uses the `barh` function with the `Horizontal` parameter. To create a vertical bar chart, use the `bar` function. To create a horizontal bar chart, use the `barh` function.
`bar3` function with a line style or marker symbol specification	Errors	Not applicable	Remove instances of code that passes a line style or marker symbol to the `bar3` function, such as `bar3(y,z,'r:*')`. In previous releases, the line style and marker symbol are ignored. Instead, specify only the color, such as `bar3(y,z,'r')`.

Functionality	Result	Use This Instead	Compatibility Considerations

`legend` function	Still runs	Not applicable	The `legend` function now always creates a legend associated with the current or specified axes. If the axes do not contain any data, then the `legend` function creates an empty legend. If axes do not exist, then the `legend` function creates axes with an empty legend.
			
			In previous releases, the `legend` function does not always create a legend. For example, it does not create a legend if there are no axes or no data in the axes. For these cases, the function returns a warning message instead. Also, the `lgd = legend` syntax (with no input arguments) returns an existing legend, but does not create one if a legend does not already exist.
			
			To query for the existence of a legend associated with a particular axes without creating a new one, use the `Legend` property of the `Axes` object. For example:
			
			`ax = gca;`
			`lgd = ax.Legend`

`subplot` function when setting the `OuterPosition` or `ActivePositionProperty` property as name-value pair arguments	Warns	`Position` parameter	Remove instances of code that sets the `OuterPosition` or `ActivePositionProperty` property as name-value pair arguments to the `subplot` function. If you want to specify the subplot position, use the `Position` property.

Functionality	Result	Use This Instead	Compatibility Considerations
`subplot` function when specifying the subplot location using both the grid position and the `Position` parameter	Warns	Either a grid position or the `Position` parameter (not both)	You cannot specify the subplot location using both a grid position and the `Position` parameter, such as `subplot(m,n,p,'Position',[x y w h])`. Instead, use either the grid position or the `Position` parameter, such as `subplot(m,n,p)` or `subplot('Position',[x y w h])`.
Behavior of charting functions when the `NextPlot` property of the `Axes` object is set to `'replacechildren'`	Still runs	Not applicable	Charting functions no longer change most axes properties when the `NextPlot` property of the `Axes` object is set to `'replacechildren'`.

Data Import and Export

Custom Datastore: Build a customized datastore

Build your own datastore for custom or proprietary data (stored in files or a data stream) using the custom datastore framework. Then, use this custom datastore to bring your data into MATLAB and leverage the MATLAB Big Data capabilities such as `tall`, MapReduce, and Hadoop. For more information, see Develop Custom Datastore.

datastore Function: Work with data stored in Windows Azure Blob Storage

You can use data stored on Windows Azure to create a datastore. Windows Azure Blob Storage (WABS) is an online file storage web service offered by Microsoft. For more information, see Introduction to Windows Azure.

To use Windows Azure with datastore, see Windows Azure Blob Storage.

datastore Function: Access Hadoop HDFS data more easily

You can access Hadoop data more easily by using the `hdfs:///` syntax:

```
ds = datastore('hdfs:///path_to_file/file.ext')
```

For example, this command creates a datastore for file `myData.txt` in folder `data`.

```
ds = datastore('hdfs:///data/myData.txt')
```

When you specify location with this syntax, the `datastore` function uses the Hadoop HDFS host name associated with the Hadoop installation that MATLAB is paired with.

FileDatastore Object: Create uniform output from datastore

For vertically concatenated data, you can design your `ReadFcn` to return a uniform output, in the form of a MATLAB table.

For example, create a `FileDastore` object and check the value of its new `UniformRead` property. This property indicates whether multiple reads of `FileDatastore` return uniform data that can be vertically concatenated. The `UniformRead` property is set to either `true` or `false`.

- If set to `false` (default), then the `readall` method returns a cell array of data where each call to the `read` method adds new entries into the cell array.
- If set to `true`, then `readall` method returns a table. Therefore, your `ReadFcn` must return vertically concatenable data else the `readall` method returns an error.

HDF5 Functions: Create datasets, groups, attributes, links, and named datatypes using non-ASCII characters

HDF5 functions support use of non-ASCII characters for HDF5 datasets, groups, attributes, links, and named datatypes. To create and access non-ASCII data, set the `'TextEncoding'` name-value pair to `'UTF-8'`. These high-level and low-level functions support the `'TextEncoding'` name-value pair:

High-Level Functions

- `h5create`
- `h5writeatt`
- `h5info`
- `h5disp`

Low-Level Functions

- `H5A.get_name`
- `H5I.get_name`
- `H5L.get_name_by_idx`
- `H5L.get_val`
- `H5R.get_name`

For more information, see Working with Non-ASCII Characters in HDF5 Files.

Web services: Skip server name verification in certificates

To disable certificate server name verification in cases where the server's certificate does not match the URI used to access it, set the `VerifyServerName` property to `false` in `matlab.net.http.HTTPOptions`.

jsonencode Function: Encode NaN and Inf as null

The `jsonencode` function encodes MATLAB NaN and Inf values in JSON as null.

Compatibility Considerations

To continue encoding NaN as `'NaN'`, call `jsonencode` with the `'ConvertInfAndNaN'` option set to `false`.

```
jsonencode(NaN,'ConvertInfAndNaN',false)
```

```
ans =

    'NaN'
```

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`ncwrite`	Still runs	`netcdf.putVar`	The `ncwrite` function applies attribute conventions `_FillValue`, `scale_factor`, and `add_offset` when writing data of all data types. If you do not want to apply attribute conventions when you write data to a netCDF variable, use `netcdf.putVar`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations

Using `load` statements inside a function to load variables from file.
 	
For example:
 	
`function loadVar()`
`  ...`
`  load data.mat x;`
`  disp(x)`
`  ...`
`end`
	Warns	none	Currently, if you use a load statement in a function to load a variable that does not exist, then MATLAB behaves as follows:
			
			1 Issues a warning `'Variable not found'`.
			2 Interprets any subsequent references to that variable, as calls to a path function (with the same name as the variable) if one exists.
			
			For example, consider a file `data.mat` that has only two variables, `a` and `b`. Write a function to load a variable `pi` from `data.mat` and include a subsequent call to `pi`.
			
			`function loadVar()`
			`  load data.mat pi; % pi does not exist in d`
			`  pi;`
			`end`
			
			Calling `loadVar` results in a warning. Then, `pi` in the next line gets resolved to `pi` on the path.
			
			`Warning: Variable 'pi' not found.`
			`> In loadVar (line 2)`
			
			`ans =`
			
			`  3.1416`
			
			In future releases, if you use a `load` statement in a function to load a variable that does not exist, then MATLAB:
			
			1 Issues the warning `'Variable not found'`.
			2 Interprets any subsequent references to that variable, as calls to a variable. Therefore, MATLAB returns the error `'Undefined function or variable'`.
			
			Attempting to load `pi` from `data.mat` and a subsequent call to `pi`, results in a warning and an error.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
			Warning: Variable 'pi' not found. > In loadVar (line 2) Undefined function or variable 'pi'. Error in loadVar (line 3) pi

Data Analysis

ischange Function: Detect abrupt changes in data

You can use the `ischange` function to find local changes in the mean, variance, or linear slope and intercept of array and table data.

islocalmin and islocalmax Functions: Detect local minima and maxima in data

To find local minima and maxima in table or array data, use the `islocalmin` and `islocalmax` functions.

rescale Function: Scale data to a specified range

You can scale data in an array to a specific range using the `rescale` function. For example, `rescale(A,1,10)` scales the entries of an array `A` to the interval [1,10].

tall Arrays: Operate on tall arrays with more functions, including fillmissing, filter, median, polyfit, and synchronize

The functions listed in this table add support for tall arrays as inputs. For a complete list of supported functions, type `methods tall`. For information on usage and limitations, type `help tall/functionName`. For example, `help tall/filter`.

`binscatter`	`plot`	`scatter`
`bounds`	`polyfit`	`split` (for tall arrays of strings)
`fillmissing`	`polyval`	`summary` (for tall timetables)
`filter`	`retime` (for tall timetables)	`synchronize` (for tall timetables)
`median`	`rmmissing`	`vertcat`

tall Array Indexing: Use subscripted assignment with tall arrays

The subscripted assignment statement `A(m,n,p,…) = B` adds support for these situations:

- The right side of the assignment statement `B` can be a tall array derived from `A`. For example, with a 3-D tall array `A = tall(ones(3,3,3))`, you can make assignments such as `A(:,:,1) = mean(A,3)`.
- The subscripts `n,p,...` can exceed the dimensions of `A`, which enables you to grow the array with subscripted assignment for any dimension other than the first (tall) dimension. For example, with a 2-D tall matrix `A = tall(rand(1000,3))`, you can turn `A` into a 3-D matrix with `A(:,:,2) = 2*A(:,:,1)`.

tallrng Function: Control random number generator used by tall arrays

Use `tallrng` in a manner similar to `rng` to control the random number generator used in certain tall array calculations.

timetable Data Container: Specify whether each variable in a timetable contains continuous or discrete data using the VariableContinuity property

To specify whether `timetable` variables represent continuous data, you can use the `VariableContinuity` property. If you specify `VariableContinuity` and call the `retime` or `synchronize` functions, then you do not need to specify a method. Instead, `retime` and `synchronize` fill in the output `timetable` variables using different default methods for continuous, step, or event data. For more information, see the `VariableContinuity` property of `timetable` or Retime and Synchronize Timetable Variables Using Different Methods.

mink and maxk Functions: Find the k smallest or largest elements in an array

Finding a specified number of smallest or largest elements in array data is now possible with the `mink` and `maxk` functions.

For example, `mink(A,5)` and `maxk(A,5)` return the 5 smallest and 5 largest elements of a vector `A`, respectively.

topkrows Function: Find the k top rows in sorted order for numeric arrays, tables, and timetables

The `topkrows` function adds support for in-memory numeric arrays, tables, and timetables. Previously, `topkrows` supported only tall arrays.

Compatibility Considerations

There are a few behavior changes when `topkrows` operates on tall arrays:

- When operating on a tall array, `topkrows` places `NaN`, `NaT`, and other missing values at the end of a descending sort. In previous releases `topkrows` placed missing values at the beginning of a descending sort.
- `topkrows` no longer accepts tall cell arrays containing only scalar numeric values as inputs. Use `cell2mat` to convert the tall cell array of scalar numeric values into a tall matrix before using `topkrows`.

App Building

App Designer: Create apps with a wide variety of 2-D and 3-D plots

Include a wider selection of 2-D and 3-D plots in your app, such as surfaces and patches. For more information, see Graphics Support in App Designer.

App Designer: Add menus to an app from the Component Library

Add menus to your apps by dragging a menu bar from the App Designer **Component Library** onto the canvas. Then add and edit your menu items by clicking the **+** buttons, pressing the **Delete** key, and typing directly on the menu labels.

App Designer: Specify input arguments when running an app

Define input arguments for your app, and access them in the `StartupFcn` callback for the `UIFigure` component. You can use the input arguments to pass data between apps or to set default values in an app. See Startup Tasks and Input Arguments in App Designer for more information.

App Designer: Add a summary, description, and screenshot for app packaging and compiling

Add a name, summary, description, and screenshot for your app. MATLAB provides this information for use in other contexts, such as:

- The **Package App** interface
- The MATLAB Compiler interface
- The file browser details on some operating systems

App Designer: Improved component Properties pane in Code View

The component **Properties** pane in **Code View** has an improved layout with richer editing capabilities. For example, toggle buttons are now available for controlling certain properties such as font weight and text alignment.

App Designer: Edit tick labels for gauges, knobs, and sliders directly in the canvas

Now you can change the tick labels for gauges, knobs, and sliders more quickly by editing them directly on the canvas.

uitree and uitreenode Functions: Create trees and tree nodes in apps

Call the `uitree` and `uitreenode` functions to display lists of items in a hierarchy within your app.

To display a hierarchy in an App Designer app, call the `uitree` and `uitreenode` functions from within a callback, such as the `StartupFcn` for the `UIFigure` component. For an example, see Use App Designer to Display Items in a Tree.

The functions only work for App Designer apps or in figures created with the `uifigure` function.

uiconfirm Function: Create modal in-app confirmation dialog boxes

Call the `uiconfirm` function to create a modal in-app dialog box that displays a set of options that you define. The function has an output argument that returns the user's selection.

The dialog box can only display in App Designer apps or in figures created with the `uifigure` function.

Toolbox Packaging: Add App Designer apps to the Apps Gallery upon toolbox installation

When you package a toolbox, MATLAB automatically detects App Designer apps (`.mlapp` files) in addition to app installer files (`.mlappinstall` files). If you package these apps or app installer files with your toolbox, they appear in the MATLAB Apps Gallery when a user installs your toolbox. For more information, see Create and Share Toolboxes.

MATLAB Online: Run App Designer apps in MATLAB Online

MATLAB Online now supports running apps created in App Designer. Only the Google Chrome browser supports this capability. If you want to build an app, you must use App Designer running on a desktop version of MATLAB.

Performance

App Designer: Load apps faster

Loading apps into App Designer is 40% to 60% faster than in R2017a. The time savings becomes more noticeable as the number of components in your app increases.

Execution Engine: Improved performance for vectorized math on CPUs with AVX2

The MATLAB execution engine improves performance of vectorized math on CPUs that support AVX2 instructions, compared to CPUs that support SSE2 or earlier versions of SIMD (Single-Instruction Multiple-Data) extensions.

Live Editor: Run live scripts with loops faster

Live scripts containing loops run significantly faster than in previous releases. In addition, typing and scrolling in live scripts is more responsive.

Hardware Support

Arduino: Wirelessly connect to Arduino boards using low-cost Bluetooth adaptors

MATLAB Support Package for Arduino Hardware enables you to wirelessly connect to and communicate with Arduino boards over Bluetooth® using low-cost Bluetooth devices such as Adafruit Bluefruit EZ-Link, HC-05, or HC-06. Instrument Control Toolbox software is required to set up Bluetooth communication.

Arduino Setup UI: Set up a connection to your Arduino board over USB, Bluetooth, or Wi-Fi

MATLAB Support Package for Arduino Hardware enables you to use `arduinosetup` guided interface to set up a connection to your Arduino board over USB, Bluetooth, or Wi-Fi.

Arduino Plug-In Detection: Discover available Arduino support and examples when plugging a compatible Arduino board

MATLAB Support Package for Arduino Hardware automatically detects compatible Arduino boards plugged into your computer and also shows available Arduino support for MATLAB and Simulink along with examples.

Advanced Software Development

MATLAB Engine API for C++: Run MATLAB code from C++ programs with object-oriented programming support and asynchronous execution

The MATLAB Engine API for C++ provides an interface between the C++ programming language and MATLAB. This API enables C++ programs to launch MATLAB, evaluate MATLAB functions with arguments, and exchange data between MATLAB and C++ programs. For more information, see MATLAB Engine API for C++.

MATLAB Engine API for C++: Pass data between C++ programs and MATLAB using MATLAB Data Array

The MATLAB Data API uses modern C++ semantics and design patterns and avoids data copies whenever possible by using MATLAB copy-on-write semantics.

Java SE 8: MATLAB support, providing improved security and access to new Java features

Java interface supports JRE version Java 8. For more information, see https://www.mathworks.com/support/sysreq/supported-language-interfaces.html.

MinGW 5.3: MATLAB support

MATLAB supports the MinGW-w64 version 5.3.0 compiler from https://mingw-w64.org for building MEX files and standalone MATLAB engine and MAT-file applications. To download the supported version, search for `MinGW` from the Add-Ons menu. If you do not install the supported version, then MATLAB displays a warning.

Microsoft Visual Studio 2017: MATLAB support for Microsoft Visual Studio 2017 Community, Professional, and Enterprise editions

MATLAB supports Microsoft Visual C++ 2017 Professional, Community, and Enterprise editions for building MEX files and standalone MATLAB engine and MAT-file applications.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Microsoft Visual C++ 2017 Professional, Community, and Enterprise editions	Windows
Added	Microsoft Visual C++ 2015 and 2013 Community and Enterprise editions in addition to continued support for the Professional edition	Windows

Support	Compiler	Platform
Added	Microsoft Visual Studio 2015 (v140) toolset installed over Visual Studio 2017	Windows
Added	MinGW-w64 version 5.3.0 compiler from https://mingw-w64.org	Windows
Added	Intel Parallel Studio XE 2017 with Microsoft Visual Studio 2017 for C, C++, and Fortran	Windows
Discontinued	MinGW-w64 version 4.9.2 compiler from TDM-GCC	Windows
Discontinued	Microsoft Visual C++ 2012 Professional	Windows
Discontinued	Microsoft Windows SDK 7.1	Windows
To be phased out	Support for GNU gcc and gfortran version 4.9 will be discontinued in a future release, at which time new versions will be supported.	Linux

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Python Version 3.6: MATLAB support

MATLAB supports the following versions of CPython:

- Version 2.7
- Version 3.4
- Version 3.5
- Version 3.6

For more information, see Install Supported Python Implementation.

Perl 5.24.1: MATLAB support

MATLAB ships with Perl version 5.24.1.

- See www.perl.org for a standard distribution of perl, perl source, and information about using perl.
- See https://metacpan.org/pod/HTML::Parser for a standard distribution of HTML::Parser, source code, and information about using HTML::Parser.
- See https://metacpan.org/pod/HTML::Tagset for a standard distribution of HTML:Tagset, source code, and information about using HTML:Tagset.

Compatibility Considerations

If you use the `perl` command on Windows platforms, see www.perl.org for information about using this version of the Perl programming language.

MATLAB Handle class method: Add a listener for an event without binding the listener to the source object

The listener handle class method adds a listener for an event without binding the listener lifecycle to the object that is the source of the event. For more information, see `handle.listener`.

Unit Testing Framework: Provide code coverage reports in the Cobertura format for improved continuous integration workflows

You can specify that the code coverage plugin returns code coverage results in the Cobertura format for use within continuous integration systems. Use the `'Producing'` name-value pair argument with the `forFolder`, `forPackage`, or `forFile` static method of the `matlab.unittest.plugins.CodeCoveragePlugin` class.

Unit Testing Framework: Generate HTML report of a test run

Create easily readable and navigable HTML reports of your test runs using the `producingHTML` static method of the `TestReportPlugin`. For more information, see `matlab.unittest.plugins.TestReportPlugin.producingHTML`.

Unit Testing Framework: Write tests as live scripts

You can write and execute script-based unit tests using Live Scripts. For more information, see Write Test Using Live Script.

Unit Testing Framework: Specify additional diagnostics to evaluate upon failures using the onFailure method

To specify additional diagnostics for the testing framework to evaluate when a test fails, use the `onFailure` method of the `matlab.unittest.TestCase` class. For more information, see `matlab.unittest.TestCase.onFailure`.

If you create custom test fixtures, you can specify additional diagnostics for failures during fixture setup and teardown routines. For more information, see `matlab.unittest.fixtures.Fixture.onFailure`.

Performance Testing Framework: Define multiple measurement boundaries in test methods

To further refine which code to measure, performance tests now support multiple measurement boundaries in a test method. Use multiple, nonnested calls to the `startMeasuring` and `stopMeasuring` methods of `matlab.perftest.TestCase` within your test methods. Measurements from multiple boundaries in the same test method are accumulated and summed. For more information, see `startMeasuring`.

Mocking Framework: Construct mocks for classes that have Abstract methods with other attributes

You can construct a mock for a class that has `Abstract` methods and other attributes. For example, you can construct a mock for a class with a method that has `Abstract` and `Static` attributes. The

mock implements the method as a concrete, `Static` method. Similarly, methods with `Abstract` and `Hidden` attributes are implemented as concrete and `Hidden` methods. For more information, see Create Mock Object.

Source Control Integration: Show differences from parent files and save copies in Git Branches

In a project under Git source control, in the Branches dialog box, you can show differences to parent files and save branches. You can save a copy of the selected files or parent files to examine added or deleted files or to test how the code ran in previous versions. For more information, see Branch and Merge with Git.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Programmatic dependence on specific diagnostic subclass from `getDiagnosticFor` method of constraint and tolerances	Still runs	Not applicable	Rework code that relies on properties or methods specific to `matlab.unittest.diagnostic.ConstraintDiagnostic` instances returned from the `getDiagnosticFor` method of `matlab.unittest.constraints` classes. In a future release, diagnostics returned from constraint and tolerance classes in the `matlab.unittest.constraints` package will be subclasses of `matlab.unittest.diagnostics.Diagnostic` and might not be instances of `matlab.unittest.diagnostic.ConstraintDiagnostic`.
Negation of `matlab.unittest.constraints.ReturnsTrue` constraint (`~ReturnsTrue`)	Warns	Not applicable	Change the logic for tests that rely on negating the `ReturnsTrue` constraint.

Functionality	Result	Use This Instead	Compatibility Considerations
Generating test suites from a class that derives from a concrete base class that defines methods that reference a `TestParameter`, `MethodSetupParameter`, or `ClassSetupParameter` not defined within the base class	Errors	Define either the: • Base class as Abstract using the class-level Abstract attribute. For example, `classdef (Abstract) MyTest < matlab.unittest.TestCase`. • Abstract parameter properties for all parameters used by methods of the class. For example, `properties (Abstract,TestParameter)`.	If your test class inherits from a concrete base class that uses a parameter that is not defined in the base class, MATLAB throws an error.
Invoking `TestRunnerPlugin` superclass methods more than once	Errors	Not applicable	Custom plugins contain methods that invoke the corresponding `TestRunnerPlugin` superclass methods exactly once. Rework any custom plugins that invoke the superclass methods more than once. Starting in R2017b, this behavior throws an error.
Setting the event `ListenAccess` or `NotifyAccesst` attributes to `immutable`.	Errors	`immutable` is not a valid value for these attributes.	In previous releases, setting these attributes to `immutable` did not cause an error.
Setting the method `Access` attribute to `immutable`.	Errors	`immutable` is not a valid value for this attribute.	In previous releases, setting this attribute to `immutable` did not cause an error.

R2017a

Desktop

Live Editor: Edit a figure interactively including title, labels, legend, and other annotations

In the Live Editor, you can add formatting and annotations to a figure interactively. To add an item go to the **Figure** tab and in the **Annotations** section, select from these available options:

-  **Title**
-  **X-Label**,  **Y-Label**
-  **Legend**
-  **Colorbar**
-  **Grid**,  **X-Grid**,  **Y-Grid**
-  **Line**,  **Arrow**,  **Text Arrow**, **Double Arrow**

For more information, see Modify Figures in Live Scripts.

Live Editor: Get suggestions for mistyped commands and variables

MATLAB suggests corrections for mistyped commands and variables in the Live Editor. If you enter an undefined command or variable name, MATLAB displays: `Did you mean:` followed by a suggested command under the error message. Press the **Fix** button to replace the mistyped command with the suggested fix.

Live Editor: Copy live script outputs to other applications

When copying from the Live Editor to other applications, in line outputs within the selection are included along with the code and text. In applications that support only plain text, the code and text is copied, but outputs are not.

You also can copy an output by right-clicking the output and selecting a copy option from the context menu

Live Editor: Hover over variables to see their current value

You can view the current value of a variable as a data tip in the Live Editor. To view the data tip, position your mouse pointer over the variable.

Data tips are enabled by default in the Live Editor. To enable or disable data tips, with a live script open in the Live Editor, go to the **View** tab, and in the **Display** section, select or deselect **Datatips**.

Add-On Explorer: Discover and install File Exchange submissions hosted on GitHub in Add-On Explorer

File Exchange submissions hosted on GitHub® are now available in the Add-On Explorer. For more information on how to get add-ons through the Add-On Explorer, see Get Add-Ons.

MATLAB Online: Use MATLAB through your web browser for teaching, learning, and convenient, lightweight access

Complement your MATLAB experience with MATLAB Online. For anytime, anywhere access, simply sign in with your MathWorks account — no download or installation required.

MATLAB Online is available with most MATLAB licenses. For more information including eligibility, visit the MATLAB Online product page.

Startup Folder Behavior Changes: Set initial working folder using new options and behaviors

The options to set the initial working folder (startup folder) have changed. For more information, see MATLAB Startup Folder.

The behavior when starting MATLAB from a shortcut icon has changed. MATLAB starts from the last working folder, except on Linux platforms or when the **Start in** value is specified on Windows platforms.

When calling MATLAB from a command prompt, the initial working folder is the terminal window folder. To use the initial working folder set in the general preferences panel, use the `-useStartupFolderPref` startup option.

```
matlab -useStartupFolderPref
```

To set the startup folder from the command prompt, use the `-sd` option. For example:

```
matlab -sd "C:\work"
```

For information about the command line options to set the initial working folder, see `matlab (Mac)`, `matlab (Windows)`, and `matlab (Linux)`.

Compatibility Considerations

If you start MATLAB from a command window and want the startup folder to be the value set in the general preferences panel, then add the `-useStartupFolderPref` startup option.

Language and Programming

string Arrays: Create string arrays using double quotes

You can create strings using double quotes, just as you can create character vectors with single quotes.

- `str = "Hello, World"` creates a string.
- `str = ["Good" "morning"]` creates a 1-by-2 string array.

For more information, see Characters and Strings.

String Functions: Return character arrays or cell arrays instead of string arrays

The data type of the output argument has changed for the `compose`, `extractAfter`, `extractBefore`, `extractBetween`, `join`, `split`, and `splitlines` functions. As of R2017a, the output data type depends on the type of the first input argument.

- If the first input argument is a string array, then the output argument is a string array.
- If the first input argument is a character vector, then the output argument is either a character vector or a cell array of character vectors.
- If the first input argument is a cell array of character vectors, then the output argument is either a character vector or a cell array of character vectors.

In R2016b, these functions always return string arrays.

missing Function: Assign missing values in core data types, including double, datetime, categorical, and string arrays

To assign missing values to elements of an array or table, use the `missing` function. For example, `catarray(3) = missing` assigns the third element of a categorical array `catarray` to a missing value.

issortedrows Function: Determine if matrix and table rows are sorted

Use the `issortedrows` function to determine if matrix or table rows are sorted. For example, `issortedrows(A,'descend')` checks if the rows of a matrix `A` are in descending order based on the elements in the first column.

sort and sortrows Functions: Specify options for sorting complex numbers and placing missing elements

Indicate how you want to sort complex numbers and place missing elements using the `sort` and `sortrows` functions with the `'ComparisonMethod'` and `'MissingPlacement'` options. For example, if `A` is a numeric vector containing complex numbers, then `sort(A,'ComparisonMethod','real','MissingPlacement','last')` sorts by the real part of each element and places `NaN` values at the end of the sort.

issorted Function: Query sort order with monotonic, strictly monotonic, strictly ascending, and strictly descending options

You can determine if an array is sorted using the `'monotonic'`, `'strictmonotonic'`, `'strictascend'`, and `'strictdescend'` options with the `issorted` function. For example, `issorted(A,'strictmonotonic')` checks if a vector `A` is strictly monotonic.

head and tail Functions: Return top or bottom rows of table or timetable

To return the top or bottom rows of a table or a timetable, you can use the `head` and `tail` functions.

table Data Containers: Use row labels when performing join, sort, and grouping operations

When you join tables, sort tables, or use grouping variables, you can specify row labels and table variables together as key variables, sorting variables, or grouping variables. You can specify them in the `innerjoin`, `join`, `outerjoin`, `rowfun`, `sortrows`, `stack`, `unstack`, and `varfun` functions. In previous releases, you could specify row labels or table variables, but not both together.

Row labels are the row names of a table or the row times of a timetable.

The `innerjoin`, `join`, and `outerjoin` functions have limitations on key variables:

- You cannot perform an inner or outer join using row labels as the left key and a table variable as the right key. To perform the inner or outer join, convert the row labels to a table variable and use the new variable as a key.
- You cannot perform an outer join using both row labels and table variables as key variables. To perform the outer join, convert the row labels to a table variable, or use the row labels as the only key variable.
- You cannot specify a table as the first input when you perform an inner or outer join on a table and a timetable. The timetable must be the first input.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations

`compose`
`extractAfter`
`extractBefore`
`extractBetween`
`join`
`split`
`splitlines`
	Still runs	Not applicable	Starting in R2017a, the output data type depends on the data type of the first input argument. In R2016b, these functions always return string arrays. For more information, see “String Functions: Return character arrays or cell arrays instead of string arrays” on page 7-4.

Functionality	Result	Use This Instead	Compatibility Considerations
`issorted` using `'rows'` option with matrix input	Still runs	`issortedrows`	To determine whether the rows of a matrix are sorted, use the `issortedrows` function. `issortedrows` provides more capability for row sort queries, such as whether the rows are in ascending or descending order.
`issorted` with timetable input	Still runs	`issortedrows`	To determine whether row times or table variables of a timetable are sorted, use the `issortedrows` function. `issortedrows` provides more capability for timetable sort queries, such as whether row times are in ascending or descending order.

`innerjoin`
`join`
`outerjoin`
`rowfun`
`sortrows`
`stack`
`unstack`
`varfun`
	Still runs	Not applicable	Starting in R2017a, you can specify row labels and table variables together as key variables, sorting variables, or grouping variables. In previous releases, you could specify row labels or table variables, but not both together. For more information, see “table Data Containers: Use row labels when performing join, sort, and grouping operations” on page 7-5.

`notebook`	Warns	Live Editor	In future releases, calling `notebook` returns an error.

Functionality	Result	Use This Instead	Compatibility Considerations
`dir`	Still runs	Not applicable	Starting in R2017a on UNIX platforms, if your current working folder is invalid and you call the `dir` function with a file or folder name, MATLAB throws an error. Your working folder becomes invalid if you delete it while it is your current working folder. In previous versions of MATLAB, if your current working folder was invalid and you called `dir` with an input, `dir` returned empty.

Graphics

heatmap Function: Visualize table or matrix data as a heatmap

To create a heatmap chart, use the `heatmap` function.

legend Function: Create legends that update when data is added to or removed from the axes

Legends now automatically update when you add or remove graphics objects from the axes. Previously, legends did not automatically add items for new graphics objects or remove items for deleted graphics objects.

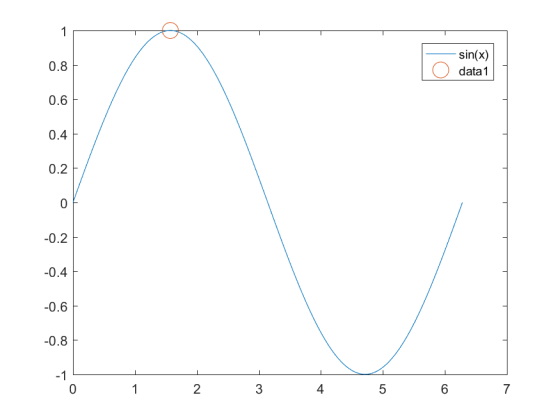
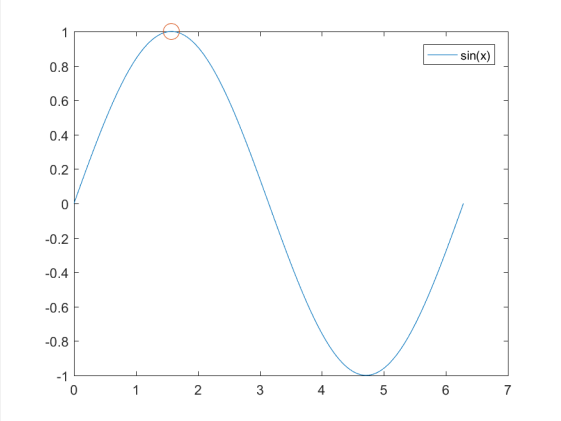
For example, this code plots a line and adds a legend. Then the code plots a second line. The legend automatically updates to include the second line.

```
plot(1:10)
legend('Line 1')
hold on
plot(11:20,'DisplayName','Line 2')
hold off
```

Compatibility Considerations

If you add or delete a graphics object after creating a legend, the number of items in the legend can differ from previous releases. For example, plot a line and create a legend. The plot displays a marker at the maximum point.

```
x = linspace(0,2*pi);
y = sin(x);
plot(x,y)
legend('sin(x)')
hold on
plot(pi/2,sin(pi/2),'o','MarkerSize',12)
hold off
```

Result in R2017a	Result in Previous Releases
The legend updates to include the marker. 	The legend does not update to include the marker. 

If you do not want the legend to update automatically, set the `AutoUpdate` property of the legend to `'off'`. Graphics objects added to the axes after the legend is created do not appear in the legend.

```
legend('sin(x)','AutoUpdate','off')
```

To change the behavior for newly created legends, set the default value of the `AutoUpdate` property. To affect all new legends, set the value on the root level. Alternatively, to affect all new legends in a particular figure, set the value on the figure level. After you set the default value, any new legends have an `AutoUpdate` property set to `'off'`.

```
fig = figure;
set(fig,'defaultLegendAutoUpdate','off')
```

Additional options for excluding specific graphics objects from a legend include:

- Specifying the graphics objects to include in the legend as an input argument to the `legend` function. For example, this code creates a legend that includes only the graphics objects referred to by `p1` and `p2`. However, graphics objects added to the axes after the legend is created do appear in the legend. Consider creating the legend after creating all the plots to avoid extra items.

  ```
  p1 = plot(1:10);
  hold on
  p2 = plot(11:20);
  p3 = plot(21:30);
  legend([p1 p2],'line 1','line 2')
  ```

- Excluding graphics objects from the legend by setting their `HandleVisibility` property to `'off'`. The `HandleVisibility` property also controls the visibility of the graphics object in the `Children` property of its parent.

  ```
  p4 = plot(31:40,'HandleVisibility','off');
  ```

- Excluding graphics objects from the legend by setting the `IconDisplayStyle` property of the underlying annotation object to `'off'`.

  ```
  p5 = plot(41:50);
  p5.Annotation.LegendInformation.IconDisplayStyle = 'off';
  ```

Categorical Plotting: Use categorical data in common plotting functions and customize axes with categorical rulers

These graphics functions accept input data of type `categorical`.

`bar`	`barh`
`plot`	`plot3`
`semilogx` (*x* values must be numeric)	`semilogy` (*y* values must be numeric)
`stem`	`stairs`
`scatter`	`scatter3`
`area`	`mesh`
`surf`	`surface`
`fill`	`fill3`
`line`	`text`

Also, you can use the new categorical ruler to customize the axes. Access the `CategoricalRuler` object through the `XAxis`, `YAxis`, or `ZAxis` property of the axes object. For example:

```
c = categorical({'red','yellow','blue'});
y = [1 2 3];
plot(c,y)
ax = gca;
ax.XAxis

ans =

  CategoricalRuler with properties:

    Categories: [blue    red    yellow]
        Limits: [blue    yellow]
    TickValues: [blue    red    yellow]

  Show all properties
```

For a complete list of properties, see `CategoricalRuler`.

histogram Function: Plot histograms of datetime and duration data

`histogram` accepts input data of type `datetime` and `duration`. Additionally, you can bin the data using units of time as the bin edges, such as `'second'`, `'hour'`, or `'week'`.

histogram Function: Sort categorical bins by bar height, and limit the number of bins displayed

`histogram` can sort categorical bins by bar height (highest and lowest), and additionally it can limit the number of bars displayed. This functionality is most useful with data sets that have a large number of categories, since it enables you to, for example, plot only the 10 largest bars or only the 15 smallest bars.

scatter Function: Create scatter plots of datetime and duration data

`scatter` and `scatter3` accept input data of type `datetime` and `duration`.

Scatter Plots: Create scatter plots with varying marker sizes faster

When you create a scatter plot and specify the marker sizes using the `SizeData` property, the scatter plot renders with improved performance.

parula Colormap: Create plots with enhanced colors

The default `parula` colormap has enhanced colors that are more perceptually uniform.

This table shows a comparison between the R2017a and R2016b versions of the parula colormap. The visual change is subtle; however, you might notice more colorful colors and smoother transitions between colors.

R2017a	
R2016b	

Compatibility Considerations

Plots that use the parula colormap maintain their overall visual appearance. However, the array of colors returned by the `parula` function is different. If your code relies on the specific RGB triplet values returned by `parula`, then you might need to update your code.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`legend` function	Still runs	Not applicable	Legends now automatically update to include new graphics objects added to the axes. Similarly, they update to exclude graphics objects deleted from the axes. For more information, see “legend Function: Create legends that update when data is added to or removed from the axes” on page 7-8.

Functionality	Result	Use This Instead	Compatibility Considerations
`parula` function	Still runs	Not applicable	Plots that use the parula colormap maintain their overall visual appearance. However, the array returned by the `parula` function contains different RGB triplet values. For more information, see “parula Colormap: Create plots with enhanced colors” on page 7-11.
`Normalization` option of `histogram` and `histogram2`	Still runs	Not applicable	The `Normalization` option of `histogram` and `histogram2` now computes the normalization using the total number of input data elements. Any data that is not binned (for example `NaN` values) or that falls outside the bin limits is still counted for the purposes of normalization. Previously, the normalization used only binned data in the calculation, which can be less than the total number of input data elements if the data contains `NaN`s or some data is outside the bin limits. For an example of how to achieve the old behavior, see Control Categorical Histogram Display.

Functionality	Result	Use This Instead	Compatibility Considerations
`area` function	Still runs	Not applicable	The `area` function now sorts the inputted data in order of increasing *x* values before plotting. Previously, the area function plotted the data in the order specified, which sometimes resulted in overlapping areas. The values of the `XData` and `YData` properties of the `Area` object are unchanged. For example, `area([3 1 2],[0 5 10])` now draws a line from `(1,5)` to `(2,10)` to `(3,0)` and fills in the area below the line. Previously, it drew a line from `(3,0)` to `(1,5)` to `(2,10)`, resulting in overlapping areas. If you want to draw filled polygons, consider using the `fill` function instead.

Data Import and Export

datastore and tabularTextDatastore Functions: Automatically detect and return date and time data in text files

The `datastore` and `tabularTextDatastore` functions detect and return date and time data as `datetime` type.

Compatibility Considerations

Previously, `datastore` and `tabularTextDatastore` functions returned date and time data as character vectors. To preserve that behavior in the `datastore` function, use:

```
ds = datastore(location,'DatetimeType','text')
```

To preserve the old behavior in the `tabularTextDatastore` function, use:

```
ds = tabularTextDatastore(location,'DatetimeType','text')
```

datastore Function: Work with data in Amazon S3 cloud storage

Create an `ImageDatastore`, `FileDatastore`, or `TabularTextDatastore` to work with data stored in the cloud using Amazon S3 (Simple Storage Service). For details, see Read Remote Data.

Import Tool: Import strings and categorical arrays interactively

The import tool now supports importing text as `string` and `categorical` data types. For more information, see **Import Tool**.

Compatibility Considerations

Previously, the Import Tool imported text data as a cell array of character vectors. To preserve that behavior, change settings in the **Text Options** field in the **Imported Data** section of the **Import** tab . Click the **Text Options** field and change the text type selection from `String Array` to `Cell Array of Character Vectors`.

detectImportOptions Function: Control import properties of fixed-width text files

Control and customize how data is imported from fixed-width text files using the `detectImportOptions` function and creating a `FixedWidthImportOptions` object.

In addition to the `SpreadsheetImportOptions` and `DelimitedTextImportOptions` objects, the `detectImportOptions` function now returns a `FixedWidthImportOptions` object. Use the `FixedWidthImportOptions` object with `readtable` to customize import options, such as:

- Import bad or missing data.
- Import only a subset of data using the `SelectedVariableNames` property.
- Manage the import of partial fields.

For more information, see `FixedWidthImportOptions`.

RESTful web services: Support for PUT and DELETE HTTP methods in webread, webwrite, and websave

To call HTTP PUT, DELETE, and PATCH methods, use the `'RequestMethod'` argument in the `weboptions` function.

save Function: Save workspace variables to a MAT-file with or without compression

Previously, the `save` command, when saving workspace variables to a MAT-file, used compression as the default (and the only) option. Now, a new option, `'-nocompression'`, allows saving of data without compression. This option is only available to use with MAT-File version `7.3`.

By default, saving a variable `myVariable` to a MAT-file in version 7.3 compresses the data:

```
save -v7.3 myFile.mat myVariable
```

To save `myVariable` without compression, use:

```
save -v7.3 -nocompression myFile.mat myVariable
```

writetable Function: Select preferred character encoding when writing to a file

Now, you can specify a character encoding of your choice by using the `'Encoding'` parameter. Previously, the `writetable` function used the system's default encoding when writing to a file and that was the only available option. For example, to set file encoding to support Japanese characters, set the `Encoding` parameter to `Shift_JIS`.

```
writetable(T, 'filename.csv', 'Encoding', 'Shift_JIS');
```

NetCDF Functions: Create variable names and attributes containing non-ASCII characters

NetCDF functions now support the use of non-ASCII characters for variable names, attribute names, and attribute values. For more information on creating NetCDF files see, NetCDF Files and `netcdf`

Webcam Support Package: GStreamer Upgrade on Linux

The MATLAB Support Package for USB Webcams Linux support now uses the GStreamer library version 1.0. It previously used version .10.

jsondecode converts JSON null values in numeric arrays to NaN

The `jsondecode` function converts JSON `null` values in numeric arrays to `NaN`. For nonnumeric arrays, this function converts a JSON `null` value to an empty `double` array (`[]`). This behavior affects the `webread` and `webwrite` functions when processing JSON content.

Compatibility Considerations

Previously, `jsondecode` decoded a JSON `null` value as an empty `double` array, which is dropped from the output. For example:

```
json = '[1, 2, null, 4]';
res = jsondecode(json) % R2016b

res =

     1
     2
     4
```

The current behavior is:

```
json = '[1, 2, null, 4]';
res = jsondecode(json) % R2017a and later

res =

     1
     2
   NaN
     4
```

To preserve the old behavior in the `jsondecode` function, that is, to remove the `NaN` values from the resulting numeric array, use the `rmmissing` function:

```
res = rmmissing(res)

res =

     1
     2
     4
```

load and fopen Functions: Use the file separator character ('\') preceding a file name to indicate that the file is in the root folder

Both the `load` and `fopen` functions take `filename` as an input argument. When specifying this `filename`, use the file separator character preceding a file name to indicate that the file is in the root folder. The file separator character is `'\'` on Windows and `'/'` on UNIX. For example, if you specify `filename` as `'\myFile.txt'`, then MATLAB searches the root folder for this file.

- If `myFile.txt` exists in the root folder, then MATLAB proceeds with loading or opening the file.
- If `myFile.txt` does not exist in the root folder, then the `load` function returns an error, and the `fopen` function returns a `-1` indicating that the file could not be opened.
- If `myFile.txt` does not exist in the root folder, but a file with the same name exists in the current folder, then MATLAB behaves in the same way as the previous step. That is, the `load` function returns an error, and the `fopen` function returns a `-1` indicating that the file could not be opened, because no such file exists in the root folder.

Compatibility Considerations

Previously, for the `load` and `fopen` functions, if you specified a `filename` as `'\myFile.txt'`, then MATLAB would look for that file in your root folder. If no such file was found in the root folder, then MATLAB would attempt to find and return the file from the current folder. However, now, if you add a file separator preceding the file name, then the `load` and `fopen` functions will only look for the file in the root folder. Therefore,

- To indicate that file is located in root folder use `load('\myFile.mat')` or `fopen('\myFile.mat')`.
- To indicate that file is located in current folder use `load('myFile.mat')` or `fopen('myFile.mat')`.

Additionally, on UNIX platforms, if you use `'/'` as the root symbol when specifying the file name, and the file is on the user path, then the path portion specified must case match. That is, MATLAB no longer case-corrects specified paths that begin with the root symbol `'/'`. For example, if a file `'/user/myFile.mat'` exists under the root folder. Then, on UNIX platforms:

- `load('/user/myFile.mat')` or `fopen('/user/myFile.mat')` works correctly.
- `load('/User/myFile.mat')` or `fopen('/User/myFile.mat')` does not work due to the case mismatch in the specified path.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Reading video files, with old file formats, on macOS platforms using `VideoReader`	Errors	Not applicable	The macOS platform no longer supports certain older video file formats. To read such files using `VideoReader`: • Open the video file using the QuickTime player. If the file is supported on macOS, the player automatically converts the file into a newer format. • Save this newly converted video file. • Use `VideoReader` to read this new converted video file.

Data Analysis

tall Arrays: Operate on tall arrays with more functions, including ismember, sort, conv, and moving statistics functions

Various functions add support for `tall` arrays as inputs.

`array2table`	`cumprod`	`movmad`	`movstd`	`sort`
`bsxfun`	`cumsum`	`movmax`	`movsum`	`sortrows`
`cell2mat`	`diff`	`movmean`	`movvar`	`table2timetable`
`conv`	`ismember`	`movmedian`	`pie` (categorical only)	`timetable`
`cummax`	`issorted`	`movmin`	`repelem`	`timetable2table`
`cummin`	`issortedrows`	`movprod`	`repmat`	`varfun`

For information about limitations of this support, type `help tall/<function>` in MATLAB, or see Functions That Support Tall Arrays (A–Z).

tall Arrays: Index tall arrays using sorted indices

Now to index tall arrays, you can use ascending or descending sorted indices. The indices can specify elements anywhere in the array, and allow for duplicates. For example:

- `T(0:2:100)`
- `T([1 2 2 5 20])`
- `T(100:-10:50)`

For more information, see Index and View Tall Array Elements.

tall Arrays: Work with out-of-memory, time-stamped data in a timetable

Convert a tall table containing a time variable into a tall `timetable` to facilitate calculations on large sets of time-stamped data. For more information, see Functions That Support Tall Arrays (A–Z).

isoutlier and filloutliers Functions: Detect and replace outliers in an array or table

To find outliers in your data, use the `isoutlier` function. To replace outliers with alternative values, use the `filloutliers` function.

smoothdata Function: Smooth noisy data in an array or table with filtering or local regression

Smoothing noisy data is now possible with the `smoothdata` function. For example, `smoothdata(A,'movmedian')` smooths data with a moving-window median.

summary Function: Calculate summary statistics and variable information in tables and timetables

To return a structure that contains a summary of a table or a timetable, use the `summary` function.

histcounts Function: Bin datetime and duration data

`histcounts` accepts input data of type `datetime` and `duration`. Also, you can bin the data using units of time as the bin edges, such as `'second'`, `'hour'`, or `'week'`.

movmad and movprod Functions: Compute moving median absolute deviation and moving product of an array

Use a sliding window to compute the moving median absolute deviation and the moving product along data in an array with the `movmad` and `movprod` functions.

bounds Function: Simultaneously determine the smallest and largest elements of an array

Find the smallest and largest elements of an array with the `bounds` function.

fillmissing Function: Replace missing data in an array or table using moving mean or moving median option

Filling missing data using a moving mean or moving median option is now available with the `fillmissing` function.

Moving Statistics Functions: Supply sample points for time-stamped and nonuniform data in moving statistics functions, such as movmean

Providing sample points that represent the location of data in an array is now possible when computing moving statistics with the functions `movmad`, `movmax`, `movmean`, `movmedian`, `movmin`, `movprod`, `movstd`, `movsum`, and `movvar`. For example, you can compute the moving-window average of data in an array `A` with respect to times in a vector `t` using `movmean(A,'SamplePoints',t)`.

prod and cumprod Functions: Ignore NaNs using 'omitnan'

Now you can exclude `NaN`s when calculating the product and cumulative product of an array with the `prod` and `cumprod` functions.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`Normalization` option of `histcounts` and `histcounts2`	Still runs	Not applicable	The `Normalization` option of `histcounts` and `histcounts2` now computes the normalization using the total number of input data elements. Any data that is not binned (for example `NaN` values) or that falls outside the bin limits is still counted for the purposes of normalization. Previously, the normalization used only binned data in the calculation, which can be less than the total number of input data elements if the data contains `NaN`s or some data is outside the bin limits. For an example of how to achieve the old behavior, see Control Categorical Histogram Display.
Support for running MATLAB MapReduce on Hadoop 1.x clusters will be removed in a future release.	Warns	Use clusters that have Hadoop 2.x or higher installed to run MATLAB MapReduce.	Migrate MATLAB MapReduce code that runs on Hadoop 1.x to Hadoop 2.x.

App Building

App Designer: Learn to build apps using an interactive tutorial

Learn how to build a simple app using an interactive tutorial that guides you through each step in the process. You can access this tutorial in the App Designer **Open** menu.

App Designer: Zoom and pan plots

Enable zooming and panning for plots in your apps using the `zoom` and `pan` functions. To enable this functionality, add buttons to your app that call `zoom` and `pan` in their callbacks. For more information, see Graphics Support in App Designer.

App Designer: Configure columns of a table to automatically fill the entire width of the table

MATLAB automatically calculates column widths so that they expand to fill all available space within the width of the table. This behavior is enabled by default, or whenever the `ColumnWidth` property of the `Table` UI component is set to `'auto'` for one or more columns.

App Designer: Manage common design-time settings using the Preferences dialog box

Specify which options are always enabled or disabled whenever you work in App Designer. For more information, see App Designer Preferences.

App Designer: Include comet, graph, and digraph visualizations in apps

Use `comet` to display comet plots, and use the graph `plot` function to display `graph` and `digraph` plots. For more information, see Graphics Support in App Designer.

App Designer: Write ButtonDownFcn callbacks for graphics objects displayed in UI axes

Create apps containing interactive plots by writing `ButtonDownFcn` callbacks for graphics objects such as `Line` or `Bar` objects. Write the callback in a separate program file that is on the MATLAB path. For example, here is a `ButtonDownFcn` callback saved as `mybuttondown.m`.

```
function mybuttondown(src,evt)
   src.Color = rand(1,3);
end
```

To create a plot line that changes color when the user clicks the line, add this code to any callback in App Designer.

```
plot(app.UIAxes,1:10,'ButtonDownFcn',@mybuttondown);
```

Note The `ButtonDownFcn` supports 2-D graphics objects that are children of the `UIAxes` (such as `Line`, `Bar`, and `Scatter`). However, `UIAxes` objects and `Figure` objects created with the `uifigure` function do not support the `ButtonDownFcn`.

App Designer: Edit table column headings directly in the canvas

Now you can edit table column headings directly in the canvas. The **Uitable Properties** panel automatically reflects your changes.

App Designer: Disable automatic resize behavior when writing SizeChangedFcn callbacks

`SizeChangedFcn` callbacks no longer execute when the automatic resize behavior is enabled. The automatic resize behavior is enabled when the **Resize components when app is resized** check box in the **UI Figure Properties** panel (Design View) is checked.

Compatibility Considerations

In previous releases, the presence of a `SizeChangedFcn` callback disabled the automatic resize behavior. The `SizeChangedFcn` callback executed whenever the container's size changed (regardless of the state of the **Resize components when app is resized** check box). Now, the automatic resize behavior must be disabled to allow the `SizeChangedFcn` to execute. Apps created in previous releases will not execute the `SizeChangedFcn` callback if the automatic resize behavior is enabled.

To disable automatic resize behavior:

1. Open your app in App Designer, and select **Design View**.
2. Click a blank area of the canvas to select the UI figure.
3. In the **UI Figure Properties** panel, deselect the **Resize components when app is resized** check box.
4. Save and run your app.

Performance

Execution Engine: Improved performance for setting MATLAB object properties

MATLAB has improved the performance of property set methods. For more information on these methods, see Property Set Methods.

save Function: Save MAT v7.3 files without compression for improved performance on some storage devices

You can improve performance on some storage devices by saving variables to MAT-File version `7.3` without compression. To save a variable (`myVariable`) without compression to a MAT-file (`myFile.mat`), use:

```
save -v7.3 -nocompression myFile.mat myVariable
```

For more information, see `save`.

memoize Function: Cache results of a function to avoid rerunning when called with the same inputs

Memoization is an optimization technique used to speed up programs by storing the results of expensive function calls and returning the cached result when the program is called with the same inputs. For more information, see `memoize`.

Scripts: Improved performance of scripts with lower script overhead

MATLAB has improved the performance of invoking scripts, especially when invoking a script from another script.

try, catch Block: Improved performance of try blocks with lower execution overhead

`try, catch` blocks that do not generate errors or throw exceptions now have lower execution overhead.

App Designer: Load apps faster

Loading apps into App Designer is 20% to 35% faster. The time savings becomes more noticeable as the number of components in your app increases.

Mathematics Functions: Various performance improvements

The following mathematics functionality shows improved performance:

- The backslash command `A\B` is faster when operating on negative definite matrices.

- The transposed backslash command `A'\B` for sparse `A` is generally faster, especially for triangular matrices.
- `unique` executes faster when the input data is sorted.
- `kron` shows improved performance with sparse matrices.
- Dimensional reduction functions (`sum`, `prod`, `any`, `all`, and so on) show improved performance when operating on the second dimension (`dim = 2`) of sparse matrices.

Hardware Support

Arduino: Read from quadrature encoders

The MATLAB Support Package for Arduino Hardware enables you to read from quadrature encoders to determine the speed, acceleration, and position of a rotating device.

Arduino: Wirelessly connect to Arduino MKR1000 board over Wi-Fi

The MATLAB Support Package for Arduino Hardware enables you to wirelessly connect and communicate to the Arduino MKR1000 board over Wi-Fi.

Advanced Software Development

Class matlab.lang.OnOffSwitchState: Represent on and off as logical values

The `matlab.lang.OnOffSwitchState` enumeration class provides better logical compatibility for properties and function arguments that have a natural “on” or “off” representation. This class enables properties that represent on/off states to accept logical values (`true`, `false`, `1`, `0`), as well as the character vectors `'on'` and `'off'` and the enumerations `on` and `off`.

Object Properties: Validate object property values by their type, size, shape, or other parameters

Class definitions for properties enable you to specify size, class, and other criteria that MATLAB uses to validate values assigned to properties. For more information, see Validate Property Values.

Validation Functions: Validate that values meet specific criteria by calling the appropriate function

Validation functions determine if values meet specific criteria and return descriptive error messages if the values do not satisfy these criteria. The primary use for validation functions is for property validation. For more information on using validation functions, see Property Validation Functions.

Name	Meaning
`mustBePositive(A)`	`A > 0`
`mustBeNonpositive(A)`	`A <= 0`
`mustBeFinite(A)`	`A` has no `NaN` and no `Inf` elements.
`mustBeNonNan(A)`	`A` has no `NaN` elements.
`mustBeNonnegative(A)`	`A >= 0`
`mustBeNegative(A)`	`A < 0`
`mustBeNonzero(A)`	`A ~= 0`
`mustBeGreaterThan(A,B)`	`A > B`
`mustBeLessThan(A,B)`	`A < B`
`mustBeGreaterThanOrEqual(A,B)`	`A >= B`
`mustBeLessThanOrEqual(A,B)`	`A <= B`
`mustBeNonempty(A)`	`A` is not empty
`mustBeNonsparse(A)`	`A` has no sparse elements.
`mustBeNumeric(A)`	`A` is numeric.
`mustBeNumericOrLogical(A)`	`A` is numeric or logical.
`mustBeReal(A)`	`A` has no imaginary part.
`mustBeInteger(A)`	`A==floor(A)`
`mustBeMember(A,B)`	`A` is an exact match for a member of `B`.

Mocking Framework: Isolate a portion of a system to test by imitating behavior of dependent components

When unit testing, you are often interested in testing a portion of a complete system isolated from dependent components. To imitate behavior of dependent components, use the mocking framework. For more information, see Mocking Framework.

Unit Testing Framework: Generate screenshots and figures during testing with ScreenshotDiagnostic and FigureDiagnostic

To record screenshots and save figures, pass `ScreenshotDiagnostic` and `FigureDiagnostic` instances to test qualifications or the `log` method of the `TestCase` class. For example, within a test, `testCase.verifyEqual(actual,expected,ScreenshotDiagnostic)` captures a screen shot when `actual` is not equal to `expected`. By default, MATLAB only records diagnostics on qualification failures. However, you can record passing diagnostics by configuring the `TestRunner` with a plugin such as the `TestReportPlugin` or `DiagnosticsRecordingPlugin`. To produce the artifacts independently from qualification failures, use the `TestCase.log` method.

You can specify where MATLAB stores these artifacts using the `ArtifactsRootFolder` property of a test runner. For more information, see `matlab.unittest.TestRunner`.

Compatibility Considerations

The `DiagnosticResult` property name has changed to `DiagnosticText` in the following classes in the `matlab.unittest.diagnostics` package: `Diagnostic`, `ConstraintDiagnostic`, `DisplayDiagnostic`, and `FunctionHandleDiagnostic`.

The `TestDiagnosticResult` and `FrameworkDiagnosticResult` properties in the `matlab.qualifications.QualificationEventData` class have been removed. They are replaced respectively by `TestDiagnosticResults` and `FrameworkDiagnosticResults` properties that contain `DiagnosticResult` objects. The properties they replace were cell arrays of character vectors.

The `DiagnosticResult` property in the `matlab.diagnostics.LoggedDiagnosticEventData` class has been removed. It is replaced by the `DiagnosticResults` property that contains `DiagnosticResult` objects. The `DiagnosticResult` property was a cell arrays of character vectors.

Unit Testing Framework: Capture screenshots and figures generated during tests using TestReportPlugin

If your tests generate screenshots and figures during tests, they are included in a test report generated with the `TestReportPlugin` class. By default, MATLAB stores only the artifacts associated with failed diagnostics. However, you can indicate that your test report includes passing diagnostics, or explicitly log the artifacts in your test using the `TestCase.log` method. For more information, see `matlab.unittest.plugins.TestReportPlugin`.

Unit Testing Framework: Control runtests function with debug, strict, and verbosity options

The `runtests` function has additional options:

- To pause test execution and enter debug mode in the event of a test failure, use the `'Debug'` option.
- To generate a qualification failure if MATLAB issues a warning during the execution of a test, use the `'Strict'` option.
- To run tests at different verbosity levels, use the `'Verbosity'` option.

For more information, see `runtests`.

Unit Testing Framework: Select tests by procedure name

You can run tests with a specified procedure name. The procedure name is different from the test element name because it does not include any class or package name or information about parameterization. In a class-based test, the procedure name is the name of the test method. In a function-based test, it is the name of the local function that contains the test. In a script-based test, it is a name generated from the test section title. For example, if a test element name for a test with parameterization is `MyTestClass/myTestMethod(param1=val1,parm2=valB)` the procedure name is `myTestMethod`.

- To select and run test elements with a specified procedure name, use the `'ProcedureName'` name-value pair with the `runtests`, `runperf`, and `testsuite` functions or the `TestSuite` suite creation methods.
- To select test elements from an existing test suite, use the `TestSuite.selectIf` method with the `HasProcedureName` selector.

Unit Testing Framework: Comparator for MATLAB tables

You can use the `TableComparator` class with the `IsEqualTo` constraint to compare table values recursively.

Compatibility Considerations

The `verifyEqual`, `assertEqual`, `assumeEqual`, and `fatalAssertEqual` methods and `IsEqualTo` constraint compare MATLAB tables using the `TableComparator` class. Prior to R2017a, they used the `ObjectComparator`, which is less strict. Therefore, it is possible that if you have tests that compare classes, test that used to pass might now fail.

Performance Testing Framework: View statistics from test measurements with the sampleSummary method

To create a `table` of summary statistics from the results for running a measurement experiment on a test suite, use the `sampleSummary` method of the `MeasurementResult` class. For more information, see `matlab.unittest.measurement.MeasurementResult.sampleSummary`.

Performance Testing Framework: Apply a function across test measurements with the samplefun method

To apply a function across the `Samples` of a `MeasurementResult` array, use the `samplefun` method. For more information, see `matlab.unittest.measurement.MeasurementResult.samplefun`.

Source Control Integration: Use Git Pull to fetch and merge in one step

You can now use Pull for Git source control integration. Pull fetches the latest changes and merges them into your current branch. Previously, you had to fetch and merge separately before you could see changes. For more information, see Pull, Push and Fetch Files with Git.

MEX builds with 64-Bit API by default

The `mex` function uses the large-array-handling API (`-largeArrayDims` option) by default. Best practice is to update your MEX source code to use this library and rebuild the MEX file. For instructions, see Upgrade MEX Files to Use 64-Bit API.

You can run existing binary MEX files without rebuilding. For more information, see Version Compatibility.

If you build MEX files without using the `mex` command options `-largeArrayDims` or `-compatibleArrayDims`, then review the table in Compatibility Considerations to avoid depending on default behavior that changes in R2017a. For information about the consequences of using the `-compatibleArrayDims` option to build MEX files, see What If I Do Not Upgrade?

The default build mode for C MEX S-functions remains `-compatibleArrayDims`.

Compatibility Considerations

This table shows the changes you must make to your `mex` command to build existing MEX files or S-functions.

Source Code	mex command — R2016b and earlier	mex command — R2017a and later
MEX file C/C++ or Fortran source code uses 32-bit API	`mex myMex.c`	`mex myMex.c -compatibleArrayDims`
	`mex myMex.c -compatibleArrayDims`	No change.
MEX file C/C++ or Fortran source code uses 64-bit API	`mex myMex.c -largeArrayDims`	Use: `mex myMex.c` Or: `mex myMex.c -largeArrayDims`

Source Code	mex command — R2016b and earlier	mex command — R2017a and later
S-function C/C++ source code uses 32-bit API	`mex sfun.c`	No change.
	`mex sfun.c -compatibleArrayDims`	No change.
S-function C/C++ source code uses 64-bit API	`mex sfun.c -largeArrayDims`	No change.
S-function Fortran source code uses 32-bit API	`mex sfun.F`	`mex sfun.F -compatibleArrayDims`
S-function Fortran source code uses 64-bit API	`mex sfun.F -largeArrayDims`	No change.

MEX files and shared libraries: Diagnostic information displayed for failure to load

Instead of displaying `Module Not Found` or `Invalid MEX-file` errors, MATLAB displays diagnostic information. For more information, see Invalid MEX File Errors and Loading Library Errors.

Java: Supports string data type

When calling a Java function, MATLAB converts `string` scalar arguments to `java.lang.String` and `string` array arguments to `java.lang.String[]`. For more information, see Pass Data to Java Methods.

The MATLAB `string` function converts `java.lang.String` scalar arguments to a `string` scalar and converts a `java.lang.String[]…[]` argument to a `string` array with the same dimensions and sizes. For more information, see Handle Data Returned from Java Methods.

Python: Supports string data type

MATLAB provides the following string data type functionality when using Python features. For more information about string data types, see Characters and Strings.

- When calling a Python function, MATLAB converts `string` scalar arguments to `py.str`. MATLAB converts `<missing>` values in `string` arguments to `py.None`. For more information, see Pass Data to Python.

- The `string` function converts `py.str` and `py.unicode` types to `string` scalar. For more information, see Handle Data Returned from Python. The `string` function does not convert objects defining `__str__` methods; for example, `py.list` or `py.dict` objects.
- The functions `pyversion` and `pyargs` accept the `string` data type for input arguments.
- `string` scalar arguments can be used to index into Python mapping types. For example, for the `py.dict` object `patient = py.dict(pyargs('name','John Doe','billing',127));`, you can display the billing value with the statement `patient{"billing"}`.

Python Version 3.3: Support discontinued

Support for Python version 3.3 is discontinued.

Compatibility Considerations

To ensure continued support for your applications, upgrade to a supported version of Python—version 3.4 or 3.5.

MATLAB ships with ActiveState Perl version 5.24 on Windows platforms

MATLAB ships with an updated version of Perl, version 5.24.

Compatibility Considerations

If you use the `perl` command on Windows platforms, refer to the ActiveState website, https://www.activestate.com, for information about this version.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2017	Windows
Added	Intel Parallel Studio XE 2017 for Fortran	macOS
Added	Xcode 8.x, as of R2016b	macOS

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Negation of `matlab.unittest.constraints.ReturnsTrue` constraint (`~ReturnsTrue`)	Still runs	Not applicable	Change the logic for tests that rely on the negating the `ReturnsTrue` constraint.
Generating test suites from a class that derives from a concrete base class that defines methods that reference a `TestParameter`, `MethodSetupParameter`, or `ClassSetupParameter` not defined within the base class	Warns	Define either the: • Base class as Abstract using the class-level Abstract attribute. For example, `classdef (Abstract) MyTest < matlab.unittest.TestCase`. • Abstract parameter properties for all parameters used by methods of the class. For example, `properties (Abstract,TestParameter)`.	In future releases, if your test class inherits from a concrete base class that uses a parameter that is not defined in the base class, MATLAB will throw an error.
Changes to `fieldnames` causes the syntax `obj.fieldnames` to issue an error with some types of objects.	Errors	Use this syntax instead: `fieldnames(obj)` To determine what properties are defined for MATLAB objects, use the `properties` function.	Use the `fieldnames` function with objects of type COM and Java.
Defining a property get method for a constant property is not supported.	Warns	Constant properties do not support the use of a get method because the value does not change by design.	In future releases, defining a property get method for a constant property will result in an error.
Linking to HTML anchor elements from custom documentation is not supported	Still runs	Not Applicable	Custom documentation pages containing links to anchors still display properly, but the browser does not scroll to the linked location.

Functionality	Result	Use This Instead	Compatibility Considerations

Built-in support for the Microsoft Source Code Control Interface, including
`cmopts`
`verctrl`
	Errors	None	MATLAB no longer includes built-in support for the Microsoft Source Code Control Interface (MSSCCI). Replace this functionality with one of these options:
			• Download and install the MATLAB integration with MSSCCI add-on.
			• Use the built-in support for Source Control Integration.

`checkin`
`checkout`
`customverctrl`
`undocheckout`
	Errors	None	Replace this functionality with one of these:
			• Built-in support for Source Control Integration
			• Source Control Software Development Kit to create a plug-in for your source control
			• MATLAB `system` function and the command-line API for your source control tool to replicate existing functionality

R2016b

Desktop

Live Editor: Pan, zoom, and rotate axes in output figures

Modify output figures interactively using the tools in the upper right corner of each axes. These tools appear when you select or hover over the axes. For more information, see Modify Output Figures.

Live Editor: Create and edit equations interactively using the equation editor

You can build an equation interactively in live scripts by selecting from a graphical display of symbols and structures. To insert a new equation, go to the **Live Editor** tab and in the **Insert** section, click **Equation**. For more information, see Insert Equations into Live Scripts.

Live Editor: Create new sections and format text quickly using autoformatting

For quick formatting in live scripts, you can use a combination of keyboard shortcuts and character sequences. For more information, including a list of supported sequences, see Autoformatting.

Live Editor: Automatically rename all functions or variables in a live script

Prevent typographical errors by automatically renaming multiple references to a function or variable within a live script. When you rename such a function or variable, a tooltip opens if there is more than one reference to that variable or function in the file. Press **Shift + Enter** to rename all instances of the function or variable.

Live Editor: Drag and drop selected code and text within a live script and between other applications

Drag selected code from a live script to and from another application, maintaining syntax highlighting and font characteristics.

Live Editor: View outputs sooner when running live scripts

The Live Editor now displays output in live scripts as it is created, instead of when the execution of a section is complete.

Command Window: View updated display for arrays, including headers indicating class, size, and shape

When displaying output, the Command Window now adds a header that includes the class, size, and shape of a variable. For example:

```
A = rand(4, 6, 'single')
```

```
A =
  4×6 single matrix
    0.8147    0.6324    0.9575    0.9572    0.4218    0.6557
    0.9058    0.0975    0.9649    0.4854    0.9157    0.0357
    0.1270    0.2785    0.1576    0.8003    0.7922    0.8491
    0.9134    0.5469    0.9706    0.1419    0.9595    0.9340
```

The Command Window displays the new header for most classes other than `double` and `char`, including:

`single`	`int8`	`uint8`	`function_handle`
`logical`	`int16`	`uint16`	`string`
`struct`	`int32`	`uint32`	
`cell`	`int64`	`uint64`	

The Command Window also displays array dimensions using a standardized format and an improved layout for integer arrays.

Product Trials: Download trials for MathWorks products using Add-On Explorer

Download and install MathWorks product trials directly from MATLAB using the Add-On Explorer. To open the Add-On Explorer, go to the **Home** tab and click the **Add-Ons**  icon.

Toolbox Packaging: Include live script examples, generate info.xml and helptoc.xml templates for custom documentation, and modify Java class path on installation

When you package a toolbox, MATLAB detects live script examples. Then, when someone installs your toolbox, the examples appear in the MATLAB Help browser.

Integrating your toolbox documentation into the MATLAB Help browser is easier with automatically generated template files. To display your toolbox documentation under **Supplemental Software** in the Help browser, edit the `info.xml` and `helptoc.xml` template files.

MATLAB automatically detects JAR files to include on the dynamic path, and you can manage these files in the Package a Toolbox dialog box.

For more information, see Create and Share Toolboxes.

Preferences: Save preferences to new default location on Mac

MATLAB now saves preferences to your user Application Support folder on Mac OS X systems. The location of the user Application Support folder is `~/Library/Application Support`.

Documentation: Find examples faster across MathWorks products

Browse examples across MathWorks products or search for examples using the search box and search filters. To access examples online, go to https://www.mathworks.com/help/examples.html. To access examples in MATLAB, go to the **Home** tab and click **Help** > **Examples**.

Documentation: Open and run examples in MATLAB

Click the **Open Script** or **Open Live Script** button in an example to open it in MATLAB. MATLAB copies the example file and all dependent files to the folder *userpath*/Examples/*examplefilename*, where *examplefilename* is the name of the example file. The *userpath* is the path returned by the `userpath` command. MATLAB then changes the current folder to the new folder. You can run the example as is or edit it before running.

Language and Programming

Functions in Scripts: Define local functions in scripts for improved code reuse and readability

MATLAB scripts, including live scripts, can contain code to define local functions. Local functions in scripts are useful for experimenting with code within a single file, as well as writing reusable code that can easily be added, modified, and deleted. For more information, see Add Functions to Scripts.

Note If your script contains local functions, you cannot run code sections (also known as code cells) individually. The  **Run Section** and  **Run and Advance** buttons are disabled. To run the script, type the saved script name in the Command Window or click  **Run**. You can run sections individually in live scripts containing local functions.

---

string Array: Manipulate, compare, and store text data efficiently

`string` arrays provide storage of text data and a set of functions for manipulating text. For example, functions such as `contains` find text in strings. The `split` and `join` functions split and join strings in string arrays. You can compare strings with relational operators such as ==, and sort them with the `sort` function.

For more information, see Characters and Strings.

timetable Data Container: Manage time-stamped tabular data with time-based indexing and synchronization

The `timetable` data container is a table that associates a time with each row. Like the `table` container, the `timetable` data container stores column-oriented data variables that have different data types and sizes, as long as the data variables have the same number of rows. To create timetables from workspace variables or from tables, use the `timetable` or `table2timetable` functions. You can resample or aggregate data in timetables with the `retime` and `synchronize` functions. You also can use any table function with timetables.

For more information, see Timetables.

timerange Function: Access all data in a specified date and time range in a timetable

You can create a subscripting variable with the `timerange` function and use that variable to subscript into a `timetable` data container.

vartype Function: Access all variables of a specified datatype in a table

You can access all table variables of a specified data type with the `vartype` function. For example, if `T` is a table, then `S = vartype('numeric'); Tnum = T(:,S)` returns `Tnum` as a table that contains only the numeric variables from `T`.

table Data Container: Reference all variables in a table with compatible types using the Variables property

You can access all table or timetable variables with compatible types using the `Variables` property. For example, if `T` is a 5-by-2 table with two numeric variables, then `T.Variables` returns a 5-by-2 numeric matrix. `T.Variables` is equivalent to the `T{:,:}` syntax.

dir Function: Search for folders and subfolders recursively

Search through folders and subfolders on the path recursively using wildcards in the path name. For more information, see `dir`.

Note The * character is now always treated as a wildcard, even on file systems that support * in file names.

Search Path: Add folders to the MATLAB search path using relative paths

When you add a folder to the MATLAB search path using a relative path, MATLAB now adds the absolute path of the folder to the search path. In previous versions of MATLAB, MATLAB added the relative path of the folder, and the path was re-resolved after every change to the current folder.

Compatibility Considerations

If your MATLAB search path contains relative paths to folders, and you rely on the automatic re-resolving of the paths when changing folders, you must update your search path. Use the Set Path dialog box or the `addpath` or `path` functions to add the absolute path of folders to the search path. For more information, see Change Folders on the Search Path.

Search Path: Add folders to the MATLAB search path using the MATLABPATH environment variable on Windows

MATLAB now adds the folders specified by the `MATLABPATH` environment variable to the MATLAB search path on Windows platforms. This action was previously only done on UNIX platforms. For more information, see Set the MATLABPATH Environment Variable.

userpath Function: Update code to use simplified userpath on UNIX

As of R2016b, the `userpath` function now only returns the first folder on the search path. In previous versions of MATLAB, `userpath` returned the first folder on the search path, appended with the folders listed in the `MATLABPATH` environment variable for UNIX platforms.

Compatibility Considerations

If you have scripts or functions that rely on the `userpath` function to return both the first folder on the search path and the paths listed in the `MATLABPATH` environment variable, then you must update your code. Use the `userpath` function to receive only the *`userpath`* folder, or use the `path` function to receive the full MATLAB search path.

regexp and regexpi Functions: Force output arguments into a cell array

You can specify the `'forceCellOutput'` argument to force the `regexp` and `regexpi` functions to return output arguments in a cell array. For example, `regexp(str,expression)` returns the indices of matches as a numeric array when `str` is a character vector or a string scalar, but `regexp(str,expression,'forceCellOutput')` returns the same numeric array within a cell.

regexptranslate Function: Replace matching patterns with escaped regular expression

You can specify the `'flexible'` argument when calling `regexptranslate` to replace matching patterns in text with an escaped regular expression that you specify. The syntax `regexptranslate('flexible',str,expression)` replaces the same text as `regexprep(str,expression,regexptranslate('escape',expression))`.

Private Functions: Visibility rules changed

As of R2016b, private functions are visible only to other functions or scripts that are in the folder immediately above the `private` subfolder (the parent folder). In previous versions of MATLAB, private functions were visible to scripts outside the parent folder if they were called by functions inside the parent folder.

Compatibility Considerations

If you have a function that calls a script outside the parent folder and that script calls a private function, your code might result in an error or change of behavior. If both the function that calls the script and the script that calls the private function are inside the parent folder, then there is no change in behavior.

If the function that calls the script and the script that calls the private function are in different folders:

- A change in behavior might occur if the script finds a different function with the same name as the private function.
- An error occurs if the script doesn't find any function with a name that matches the private function name.

Place scripts that need access to private functions to the folder immediately above the `private` subfolder.

Message Identifiers: Update code to use modified error message identifiers

The error message identifier `MATLAB:datatypes:valueOutsideValidRange` has changed.

Compatibility Considerations

If you have scripts or functions that use the `MATLAB:datatypes:valueOutsideValidRange` error message identifier, you must update the code to use one or more of the new identifiers listed here.

Typically, you might use error message identifiers in code that uses a try/catch statement and performs an action based on a specific error identifier.

- `MATLAB:datatypes:OutsideRangeFiniteGreater0`
- `MATLAB:datatypes:OutsideRangeIntegerGreater0`
- `MATLAB:datatypes:OutsideRangeNaN0to1`
- `MATLAB:datatypes:OutsideRangeValidNumber`
- `MATLAB:datatypes:OutsideRange0to1`

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Calling `nargin`, `nargout`, or `inputname` from within a script	Errors	Move calls to `nargin`, `nargout`, and `inputname` into the relevant function file. If you need the results in a script, call the functions and store the results in a variable. Then use the variable in the script.	If you call `nargin`, `nargout`, or `inputname` from within a script, MATLAB throws an error.
Visibility of local functions	Errors	Store the function handle to the local function in a variable and use that instead of the call to the local function. For example, assign `fcn = @myLocalFcn` and update the script to use the variable `fcn` instead of calling `myLocalFcn` directly.	Local functions are visible only to other functions in the same file. Previously, if a function file that defines local functions invoked a script, that script could call those local functions.
Scripts that call private functions, but do not reside in the folder immediately above the `private` subfolder	Errors	Store the function handle to the private function in a variable and use that instead of the call to the private function. For example, assign `fcn = @myPrivateFcn` and update the script to use the variable `fcn` instead of calling `myPrivateFcn` directly.	Private functions are visible only to other functions or scripts in the folder immediately above the `private` subfolder. For more information, see “Private Functions: Visibility rules changed” on page 8-7.
Linear indexing to expand `struct` arrays and `cell` arrays with brace notation	Errors	To use linear indexing to expand a multidimensional `struct` array or `cell` array that uses brace notation, first reshape the array to a 1-by-N or N-by-1 array. Alternatively, expand the array using multidimensional indexing.	Previously, linear indexing to expand the bounds of `struct` arrays or `cell` arrays using braces would automatically reshape the array to an N-by-1 array.

Functionality	Result	Use This Instead	Compatibility Considerations
Deletion of table rows or variables by subscripting into table and assigning empty quotation marks (`''`)	Errors	To delete table rows or variables, subscript into the table row or variable, and assign empty square brackets (`[]`) instead of empty quotation marks (`''`). For example, to delete the first row of table `T`, use `T(1,:) = []` syntax.	Previously, an assignment using empty quotation marks when subscripting into a table would delete table rows or variables.
Deletion of `duration` or `calendarDuration` elements by subscripting into array and assigning empty quotation marks (`''`)	Errors	To delete `duration` or `calendarDuration` elements, subscript into the array, and assign empty square brackets (`[]`) instead of empty quotation marks (`''`). For example, to delete the second element of `duration` array `D`, use `D(2) = []` syntax.	Previously, an assignment using empty quotation marks when subscripting into a `duration` or `calendarDuration` array would delete elements of such arrays.
Deletion of `categorical` elements by subscripting into an array and assigning empty quotation marks (`''`)	Still runs	To delete `categorical` elements, subscript into the array, and assign empty square brackets (`[]`) instead of empty quotation marks (`''`). For example, to delete the second element of `categorical` array `C`, use `C(2) = []` syntax.	An assignment using empty quotation marks when subscripting into a `categorical` array now assigns `<undefined>` elements to the array. Previously, the assignment deleted elements.
Deletion of `datetime` elements by subscripting into an array and assigning empty quotation marks (`''`)	Still runs	To delete `datetime` elements, subscript into the array, and assign empty square brackets (`[]`) instead of empty quotation marks (`''`). For example, to delete the second element of `datetime` array `D`, use `D(2) = []` syntax.	An assignment using empty quotation marks when subscripting into a `datetime` array now assigns `NaT` values to the array. Previously, the assignment deleted elements.

Mathematics

Implicit Expansion: Apply element-wise operations and functions to arrays with automatic expansion of dimensions of length 1

Implicit expansion is a generalization of scalar expansion. With scalar expansion, a scalar expands to be the same size as another array to facilitate element-wise operations. With implicit expansion, the element-wise operators and functions listed here can implicitly expand their inputs to be the same size, as long as the arrays have compatible sizes. Two arrays have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of them is 1. See Compatible Array Sizes for Basic Operations and Array vs. Matrix Operations for more information.

- **Element-wise arithmetic operators** — `+`, `-`, `.*`, `.^`, `./`, `.\`
- **Relational operators** — `<`, `<=`, `>`, `>=`, `==`, `~=`
- **Logical operators** — `&`, `|`, `xor`
- **Bit-wise functions** — `bitand`, `bitor`, `bitxor`
- **Elementary math functions** — `max`, `min`, `mod`, `rem`, `hypot`, `atan2`, `atan2d`

For example, you can calculate the mean of each column in a matrix `A`, and then subtract the vector of mean values from each column with `A - mean(A)`.

Previously, this functionality was available via the `bsxfun` function. It is now recommended that you replace most uses of `bsxfun` with direct calls to the functions and operators that support implicit expansion. Compared to using `bsxfun`, implicit expansion offers faster speed of execution, better memory usage, and improved readability of code.

Compatibility Considerations

If your code uses any of the element-wise operators or functions listed above and relies on the errors that MATLAB returns from mismatched sizes, particularly within a `try/catch` block, then your code might no longer catch those errors. This change is because some combinations of input sizes that previously caused errors are now valid.

For example, in older releases of MATLAB, you could not add a row and a column vector, but those operands are now valid for addition with implicit expansion. In other words, an expression like `[1 2] + [1; 2]` previously returned a size mismatch error, but now it executes.

This compatibility consideration does not apply to any uses of `bsxfun`.

graph and digraph Objects: Compute graph isomorphism, biconnected components, cut vertices, and node condensation

Graph objects created using either `graph` or `digraph` support these functions:

- `isomorphism` — Compute graph isomorphism equivalence relation between two graphs.
- `isisomorphic` — Determine if two graphs are isomorphic.

Graph objects created using `graph` support these functions:

- `biconncomp` — Compute biconnected components of undirected graph.
- `bctree` — Compute block-cut tree of undirected graph.

Graph objects created using `digraph` support these functions:

- `condensation` — Represent each strongly connected component in a directed graph using a single node.

graph and digraph Objects: Visualize graphs and networks in 3-D

The `GraphPlot` object returned by plotting a `graph` or `digraph` now contains a `ZData` property. If `ZData` is nonempty, then it specifies *z*-coordinates for the graph nodes.

Additionally, new layout methods are available for `'force3'` and `'subspace3'`, which automatically calculate values for `XData`, `YData`, and `ZData` to display the graph in three dimensions.

digraph Object: Reverse edge directions in a directed graph using the flipedge function

The `flipedge` function reverses the direction of directed edges in a graph.

conv2 Function: Compute 2-D convolutions with improved performance

The `conv2` function performs faster when using the syntax `conv2(u,v,A)`, where `u` and `v` are vectors and `A` is a matrix.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`circshift` behavior change for row vectors	Still runs	To preserve previous behavior when operating on row vectors with a scalar shift factor, specify 1 for the dimension argument. For example, use `circshift([1 2 3 4],2,1)`, where 2 is the shift factor and 1 is the dimension.	`circshift` now shifts elements along row vectors when provided a scalar shift factor. For example, `circshift([1 2 3 4],2)` returns the vector `[3 4 1 2]`. Previously, `circshift` returned the original row vector with no shifting.

Graphics

Date and Time Plotting: Use datetime or duration data in common plotting functions, and customize plots with datetime and duration rulers

To plot datetime or duration data, you can use these functions.

`bar`	`barh`
`plot`	`plot3`
`semilogx` (*x* values must be numeric)	`semilogy` (*y* values must be numeric)
`stem`	`stairs`
`area`	`mesh`
`surf`	`surface`
`fill`	`fill3`
`line`	`text`

Also, you can use the new datetime and duration ruler objects to customize the axes. Access the datetime or duration ruler object through the axes object. For example:

```
d = duration(1,30:33,0);
y = [3 5 2 3];
plot(d,y)
ax = gca;
ax.XAxis

ans =

  DurationRuler with properties:

            Limits: [01:29:55    01:33:04]
        TickValues: [01:30:00    01:30:30    01:31:00    01:31:30    01:32:00    01:32:29    01
    TickLabelFormat: 'hh:mm:ss'

  Show all properties
```

For a list of object properties, see Datetime Ruler Properties and Duration Ruler Properties.

Compatibility Considerations

When plotting datetime or duration data, consider these changes:

- The `xlim`, `ylim`, `zlim`, and `axis` functions no longer accept double values to set the limits for an axis with datetime or duration values.
- The `axis` function no longer supports setting the axis limits for axes with mixed data types (numeric, datetime, or duration). Use the `xlim`, `ylim`, and `zlim` functions instead.
- The `XLim`, `YLim`, `ZLim`, `XTick`, `YTick`, and `ZTick` properties of the axes object now contain datetime or duration values instead of double values. To change the limits or tick values, specify datetime or duration values instead of double values.

- The `XData`, `YData`, and `ZData` properties now contain datetime or duration values instead of double values. To change the data values, specify datetime or duration values instead of double values.
- Combining plots that mix different data types along a single axis results in an error. When combining plots, use only one data type per axis.

polarscatter and polarhistogram Functions: Create scatter and histogram plots in polar coordinates

To create scatter plots in polar coordinates, use the `polarscatter` function. Similarly, you can create histograms in polar coordinates using the `polarhistogram` function. The `polarhistogram` function supersedes the `rose` function.

fimplicit and fimplicit3 Functions: Plot implicit functions of the form f(x,y) = 0 and f(x,y,z) = 0

To plot implicit functions of the form $f(x,y) = 0$, use the `fimplicit` function. To plot implicit functions of the form $f(x,y,z) = 0$, use the `fimplicit3` function.

Tick Formatting Functions: Customize text, position, rotation, and format for axis tick labels

You can customize the appearance of the tick values and labels along each axis using tick formatting functions. Use the *x*-axis, *y*-axis, and *z*-axis versions when plotting in Cartesian coordinates. Use the *r*-axis and *theta*-axis versions when plotting in polar coordinates.

This table lists the new functions.

Functions	Description

`xticks`
`yticks`
`zticks`
`rticks`
`thetaticks`
	Customize the locations of the tick marks along each axis.

`xticklabels`
`yticklabels`
`zticklabels`
`rticklabels`
`thetaticklabels`
	Customize the tick labels along each axis.

`xtickformat`
`ytickformat`
`ztickformat`
`rtickformat`
`thetatickformat`
	Format the tick labels along each axis, such as controlling the number of decimal values shown or displaying the values as currency values.

`xtickangle`
`ytickangle`
`ztickangle`
`rtickangle`
	Rotate the tick labels along each axis.

errorbar Function: Create vertical and horizontal error bars and control hat width

To plot both vertical and horizontal error bars, use the `errorbar` function. Also, you can control the width of the caps at each end of the error bars by setting the `CapSize` property of the errorbar object.

```
x = 1:10;
y = sin(x);
err = std(y)*ones(size(x));
e = errorbar(x,y,err,'horizontal')
e.CapSize = 10;
```

These new errorbar object properties control the lengths of the error bars in each direction:

- `XNegativeDelta` and `XNegativeDeltaSource` — Length of left side of horizontal error bars
- `XPositiveDelta` and `XPositiveDeltaSource` — Length of right side of horizontal error bars
- `YNegativeDelta` and `YNegativeDeltaSource` — Length of lower portions of vertical error bars
- `YPositiveDelta` and `YPositiveDeltaSource` — Length of upper portions of vertical error bars

Compatibility Considerations

These errorbar object properties still run, but have been replaced with new properties.

Replaced Property	Use This Property Instead
`LData`	`YNegativeDelta`
`UData`	`YPositiveDelta`
`LDataSource`	`YNegativeDeltaSource`
`UDataSource`	`YPositiveDeltaSource`

plot Function: Control location and frequency of markers with the MarkerIndices property

To control the data points at which markers display, use the `MarkerIndices` property of lines. For example, this code displays a marker every 10 data points.

```
x = linspace(0,10,500);
y = sin(x);
plot(x,y,'-o','MarkerIndices',1:10:length(y))
```

histogram and histogram2 Functions: Create a histogram from precomputed bin counts

Plot histograms using `histogram` and `histogram2` by directly passing in the bin counts. When you use this method, `histogram` and `histogram2` do not do any data binning.

Contour Plots: Generate contour matrix with improved performance

Commands that require computing the contour matrix now perform faster, such as:

- Returning the contour matrix as an output argument from the `contour` or `contourf` functions
- Querying the value of the `ContourMatrix` property of a contour object

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`legend` function	Still runs	Not applicable	The behavior of `legend` has changed. The `legend` function now reuses an existing legend. Thus, recalling the `legend` function does not reset properties such as the location or orientation. Previously, the `legend` function deleted any existing legends and recreated a new one.
`legend('show')`	Still runs	Not applicable	If axes exist and contain no graphics objects, then `legend('show')` creates an empty legend. Previously, it returned a warning.
`rose` function	Still runs	`polarhistogram` function	Replace all instances of the `rose` function with the `polarhistogram` function.
`xlim`, `ylim`, `zlim`, and `axis` functions — Specifying limits as double values for an axis with datetime or duration values	Errors	Not applicable	When specifying limits for an axis with datetime or duration values, the `xlim`, `ylim`, `zlim`, and `axis` functions no longer accept double values. Specify datetime or duration values instead.
`axis` function — Setting limits with different data types	Errors	`xlim`, `ylim`, and `zlim` functions	Setting limits of different data types using the `axis` function results in an error. Use the `xlim`, `ylim`, and `zlim` functions instead.

Functionality	Result	Use This Instead	Compatibility Considerations
`XTick`, `YTick`, `ZTick`, `XLim`, `YLim`, and `ZLim` properties of the axes object — Specifying tick values or limits as double values for an axis with datetime or duration values	Errors	Not applicable	When specifying tick values or limits for an axis with datetime or duration values, the `XTick`, `YTick`, `ZTick`, `XLim`, `YLim`, and `ZLim` properties of the axes object no longer accept double values. Specify datetime or duration values instead.
`XData`, `YData`, and `ZData` properties of graphics objects	Still runs	Not applicable	When plotting datetime or duration data, the `XData`, `YData`, and `ZData` properties of graphics object now contain datetime or duration values instead of double values. To change the data values, specify datetime or duration values instead of double values.
Combining multiple plots that mix different data types (numeric, datetime, or duration data) along a single axis	Errors	One data type per axis	When combining plots, use only one data type per axis.
`LData` errorbar property	Still runs	`YNegativeDelta` errorbar property	Replace instances of `LData` with `YNegativeDelta`.
`LDataSource` errorbar property	Still runs	`YNegativeDeltaSource` errorbar property	Replace instances of `LDataSource` with `YNegativeDeltaSource`.
`UData` errorbar property	Still runs	`YPositiveDelta` errorbar property	Replace instances of `UData` with `YPositiveDelta`.
`UDataSource` errorbar property	Still runs	`YPositiveDeltaSource` errorbar property	Replace instances of `UDataSource` with `YPositiveDeltaSource`.
Default value of `Type` property for polar axes objects	Still runs	New default value of `'polaraxes'`	The `Type` property for polar axes objects now has a default value of `'polaraxes'`. In previous releases the value was `'axes'`. Update code to use the new value.

Functionality	Result	Use This Instead	Compatibility Considerations
Default value of `AlignVertexCenters` property for errorbar objects	Still runs	Not applicable	The `AlignVertexCenters` property for errorbar objects now has a default value of `'on'`. In previous releases the value was `'off'`. Update code to use the new value.

Data Import and Export

readtable Function: Automatically detect and return date and time data in text and spreadsheet files

The `readtable` function detects and returns date and time data as `datetime` type.

Compatibility Considerations

Previously, `readtable` function returned date and time data as character vectors. To preserve that behavior use `T = readtable(filename,'DatetimeType','text')`.

detectImportOptions Function: Detect layout of text and Excel files and customize import options to readtable

You can control and customize how data is imported from text and spreadsheet files using the `detectImportOptions` function.

The `detectImportOptions` function returns a `SpreadsheetImportOptions` object for spreadsheet files and a `DelimitedTextImportOptions` object for text files. Use these objects with `readtable` to customize import options, such as:

- Import bad or missing data.
- Import only a subset of data using the `SelectedVariableNames` property.
- Customize the decimal separator and thousands separator for numeric data using the `DecimalSeparator` and the `ThousandsSeparator` properties.
- Import data as `logical` variables from a text file.

For more information on properties of objects created by the `detectImportOptions` function, see `SpreadsheetImportOptions` and `DelimitedTextImportOptions`.

VideoReader Object: Read video frames more quickly from MP4 and MOV files on Windows systems

The `VideoReader` object now supports full rate reading for HD Videos.

imageDatastore Function: Read batches of images for faster processing in machine learning and computer vision applications

Now read minibatches from the `ImageDatastore` object using the `ReadSize` property. See `imageDatastore` function.

TallDatastore Object: Efficiently retrieve preprocessed and cleaned-up data saved using the write function of the MATLAB tall arrays

To speed up working with large data files use the `TallDatastore` object to recreate MATLAB `tall` arrays from files written to disk by the `write` function. You can use the `TallDatastore` object to efficiently load and work with data that has been previously preprocessed and cleaned up.

jsondecode, jsonencode Functions: Encode and decode structured data in JSON-formatted text

Use the `jsondecode` function to parse JSON text and convert it into MATLAB data types.

Use the `jsonencode` function to encode MATLAB data into JSON-formatted text.

writetable Function: Support for writing missing fields of a table to a spreadsheet file

You now can write blank cells in place of <missing> values using the `writetable` function.

readtable, textscan, tabularTextDatastore, and spreadsheetTextDatastore Functions: Support string data type using the 'TextType' parameter

The functions `readtable`, `textscan`, `tabularTextDatastore`, and `spreadsheetDatastore` support the importing of text as a `string` data type. For example, to read any text variable in the file `myfile.txt` as a `string` array, use `T = readtable('myfile.txt','TextType','string')`. For more information on `string` arrays, see Characters and Strings.

weboptions Function: Create custom HTTP headers and specify HTTPS certificates

To create custom HTTP header fields, use the `'HeaderFields'` argument in the `weboptions` function.

To specify HTTPS certificates, use the `'CertificateFilename'` argument in the `weboptions` function.

Scientific File Format Libraries: CDF Library upgraded to version 3.6.1

The CDF library has been upgraded to version 3.6.1.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`movie2avi`	Errors	`VideoWriter`	Remove all instances of `movie2avi`. Write to AVI files with `VideoWriter`.

Functionality	Result	Use This Instead	Compatibility Considerations
`fopen` — Specifying the `filename` argument as a character column vector	Errors	Specify `filename` as a character row vector or a string scalar.	When specifying the `filename` argument, the `fopen` function no longer accepts a character column vector. Specify `filename` as a character row vector or a string scalar.

Data Analysis

tall Arrays: Manipulate and analyze data that is too big to fit in memory

Tall arrays provide a way to work naturally with out-of-memory data. Many MATLAB operations and functions work the same way with tall arrays as they do with in-memory arrays. However, tall arrays are not stored directly in memory and are evaluated by request using `gather`. MATLAB automatically optimizes the queued calculations by minimizing the number of passes through the data. For more information, see Tall Arrays.

You can create tall numeric arrays, cell arrays, categoricals, strings, datetimes, durations, or calendar durations, and you can use any of these tall types as variables in a tall table. For more information, see Functions That Support Tall Arrays (A–Z).

Missing Data Functions: Find, fill, and remove missing data from arrays or tables with ismissing, standardizeMissing, fillmissing, and rmmissing

You can clean missing data from an array or table using the following functions:

- `ismissing` — Find elements with missing values.
- `standardizeMissing` — Inserts missing value indicators.
- `fillmissing` — Replaces missing values with a specified method, such as linear interpolation or a constant value.
- `rmmissing` — Removes missing entries.

Cumulative Functions: Ignore NaNs using 'omitnan' in cumsum, cummin, and cummax

You now can exclude NaNs when calculating cumulative statistics with the `cumsum`, `cummin`, and `cummax` functions.

discretize Function: Discretize datetime and duration arrays to separate time-stamped data into regular time intervals

The `discretize` function accepts `datetime` and `duration` arrays as inputs.

Constrained Pan and Zoom: Pan or zoom in a single dimension for 2-D and 3-D plots

You can pan or zoom in a single dimension using the new constrained options.

- To pan in a single dimension, first enable panning mode; for example, select the pan icon in the figure toolbar. Then, click and drag the tick marks or tick labels of the *x*-axis, *y*-axis, or *z*-axis to pan in that direction. The  visual indicates that you are performing a constrained pan

operation. Alternatively, right-click the axes and set the pan constraint using the context menu. See `pan` for information about the programmatic alternative.

- To zoom in a single dimension, first enable zooming mode; for example, select one of the zoom icons in the figure toolbar. For 2-D plots, click and drag the mouse in approximately the horizontal or vertical direction. The  visual indicates that you are performing a constrained zoom operation. Alternatively, right-click the axes and set the zoom constraint using the context menu. For 3-D plots, use the context menu. See `zoom` for information about the programmatic alternative.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`getAxesZoomMotion` and `setAxesZoomMotion` object functions for zoom objects	Still runs	`getAxesZoomConstraint` and `setAxesZoomConstraint` object functions for zoom objects	Replaces all instances of `getAxesZoomMotion` with `getAxesZoomConstraint`. Replace all instances of `setAxesZoomMotion` with `setAxesZoomConstraint`.
`getAxesPanMotion` and `setAxesPanMotion` object functions for pan objects	Still runs	`getAxesPanConstraint` and `setAxesPanConstraint` object functions for pan objects	Replaces all instances of `getAxesPanMotion` with `getAxesPanConstraint`. Replace all instances of `setAxesPanMotion` with `setAxesPanConstraint`.

App Building

App Designer: Include common 2-D plots in apps, such as area, bar, contour, histogram, yyaxis, and function plots

Now you can display most 2-D plots in your app. For more information, see Graphics Support in App Designer.

App Designer: Create legends for 2-D plots in apps

Add legends and color bars to your plots in App Designer. For more information, see Graphics Support in App Designer.

App Designer: Embed tabular displays using uitable in apps

Use App Designer to display tabular data in your app. Drag a **Table** component onto the canvas and edit the `Data` property in the code view to display your data.

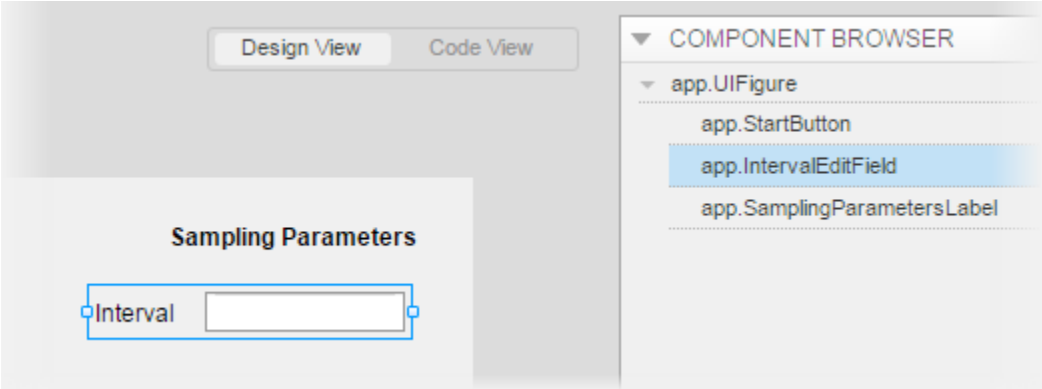
If your `Table` component displays a large number of rows and columns, you might notice a brief delay and a progress indicator as you scroll through the table in your running app. The delay occurs as MATLAB refreshes your data.

App Designer: Create callback functions that can be shared across multiple components

Write a callback function and share it among multiple components in your app. Right-click on a component in the canvas or the **Component Browser** and select **Callbacks** from the context menu. The Add Callback Function dialog box allows you to specify a new callback name or select from a list of existing callbacks.

App Designer: Add, position, and resize labeled components, and create components with names derived from labels

- The resizing behavior of labeled components is improved.
- The names of labeled components are now based on the label text, making it easier to identify the components in the **Component Browser** and code view.
- Ungrouped labels are accessible in the App Designer **Component Browser** by default.



App Designer: Edit spinners and numeric edit field values directly in the canvas and duplicate components into different parents

- Now you can edit the default value for a numeric edit field or spinner directly on the canvas.
- Duplicate any component into a different parent container by pressing **Ctrl**, selecting the component, and dragging the duplicated component into the target container.

App Designer: Learn App Designer workflow by showing tips in the code view

Click the **Show Tips** button in the App Designer code view to see a brief overview of the coding workflow. The tips point to specific areas of App Designer and guide you through the basic steps of creating an app.

App Designer Components: Position property uses 1-based origin

The `Position` property of App Designer components uses a 1-based origin in pixel units. In R2016a, App Designer components used a 0-based origin.

Compatibility Considerations

When you use R2016b to run an App Designer app created in R2016a, the UI components are shifted down and to the left by one pixel. You can use App Designer to correct the issue by opening and saving the app in R2016b before running it.

Similarly, when you use R2016a to run an App Designer app created in R2016b, the UI components are shifted up and to the right by one pixel.

Edit Field Component: Use the ValueChangingFcn property to execute a callback when users edit the value in the UI

`ValueChangingFcn` is a new property for the `EditField` component. When you specify a callback for the `ValueChangingFcn` property, the callback executes as the user edits the value at run time.

Positioning Containers Programmatically: Access the outer bounds and drawable area of containers such as panels and button groups

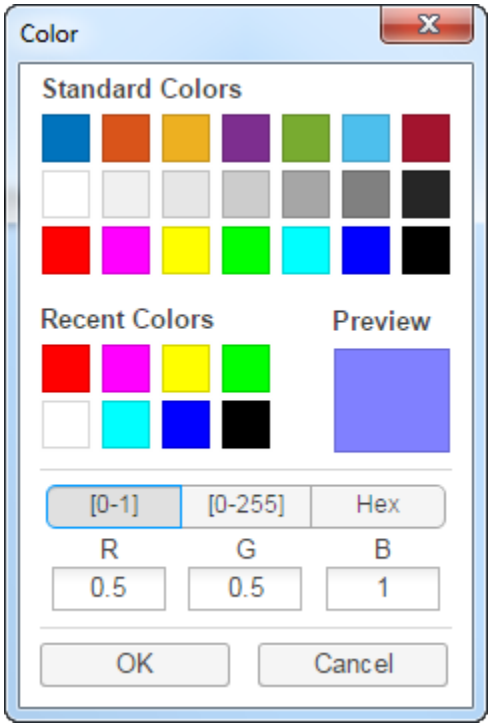
These containers include an `OuterPosition` property for accessing the location and size of their outer bounds. They also include an `InnerPosition` property for accessing the location and size of their drawable area.

- Panels — See Uipanel Properties for more information.
- Button groups — See Uibuttongroup Properties for more information.
- Tab Groups — See Uitabgroup Properties for more information.

uisetcolor Function: Use the redesigned color picker to access recent colors and specify RGB values

The color picker that the `uisetcolor` function displays has been redesigned to include the following options:

- A palette of standard colors at the top
- A palette of recently selected colors in the center
- RGB input fields that update the color preview as you edit the values



App Packaging: Automatically include dependent files when packaging apps

You no longer need to include files accessed using standard file format functions or low-level I/O functions. The Package App dialog box automatically includes:

- Files that are accessed using standard file format functions. These dependent files include text, spreadsheets, images, audio, and video files. Scientific data formats are the only standard file formats that are not supported.
- Files that are accessed using low-level I/O functions.

For a list of supported functions, see Standard File Formats. For a list of supported I/O functions, see Low-Level File I/O.

Performance

Graphics Display: Render plots with large numbers of markers faster using less memory

Graphics that contain large numbers of markers have improved performance and use less memory.

Execution Engine: Execute tight loops with scalar math faster

Performance is improved for MATLAB code that is dominated by tight loops, straightforward indexing, and simple math.

Execution Engine: Construct objects faster

Performance is improved for creating simple objects in MATLAB. Improvements apply to MATLAB and MathWorks Toolbox objects (for example, `datetime`, `inputParser`) and user-authored objects. For more information, see “Object Constructors: Construct objects faster with certain constraints” on page 8-30.

Hardware Support

iPhone and Android Sensors: Log data from mobile sensors on the MathWorks Cloud over a Wi-Fi or cellular network

The MATLAB Support Package for Android Sensors and the MATLAB Support Package for Apple iOS Sensors now work on the MathWorks Cloud.

For information about how to connect your Android device to the Cloud, see the “Use MATLAB on the MathWorks Cloud” subsection in “Set Up and Connect to Android Device” in the MATLAB Support Package for Android Sensors documentation.

For information about how to connect your Apple iOS device to the Cloud, see the “Use MATLAB on the MathWorks Cloud” subsection in “Set Up and Connect to Apple iOS Device” in the MATLAB Support Package for Apple iOS Sensors documentation.

Arduino: Write to shift registers

Write to shift registers using the MATLAB Support Package for Arduino Hardware.

Raspberry Pi: Generate PWM signals and control servo motors from GPIO pins on Raspberry Pi

You can use MATLAB commands to create pulse-width-modulated (PWM) signals and control servo motors from GPIO pins on the Raspberry Pi hardware. For more information, see MATLAB Support Package for Raspberry Pi Hardware.

Raspberry Pi: Support for Raspberry Pi 3 Model B

You can use the MATLAB Support Package for Raspberry Pi Hardware with the Raspberry Pi 3 Model B hardware board.

Raspberry Pi: Read from sensors and write to the LED matrix on a Raspberry Pi Sense HAT

You can use the MATLAB Support Package for Raspberry Pi Hardware with Raspberry Pi Sense HAT board to read from the sensors and joystick, and write to the LED matrix.

The components that you can communicate with from the support package are:

- Humidity sensor - Read the relative humidity and the ambient temperature.
- Pressure sensor - Read the barometric air pressure and the ambient temperature.
- IMU sensor
  - Accelerometer - Read the linear acceleration along the x, y, and z axes.
  - Gyroscope - Read the angular velocity along the x, y, and z axes
  - Magnetometer - Read the magnetic field along the x, y, and z axes.

- Joystick - Read the status of the joystick.
- LED matrix - Write to a pixel, display an image, or display a message on the 8x8 RGB LED matrix.

Raspberry Pi: Run Linux and file management commands faster

Run Linux and file management commands faster with performance improvements to the `system`, `getFile`, `putFile`, and `deleteFile` functions in the MATLAB Support Package for Raspberry Pi Hardware.

Advanced Software Development

matlab.net.http Object: Access HTTP services with low-level protocol control

For more information, see HTTP Interface.

MATLAB Engine API for Java: Run MATLAB code from Java programs

Call MATLAB functions and evaluate MATLAB statements from Java programs synchronously or asynchronously. Java programs can pass data to and get data from the MATLAB base workspace. For more information, see MATLAB Engine API for Java.

matlab.mixin.SetGet: Allow inexact property names by default in calls to set and get

Classes derived from `matlab.mixin.SetGet` support inexact property name matching by default in calls to `set` and `get`. To require exact property name matching, derive your class from the `matlab.mixin.SetGetExactNames` class.

matlab.mixin.SetGetExactNames: Require exact property names in calls to set and get

Classes derived from `matlab.mixin.SetGetExactNames` require case-sensitive, exact property name matches in calls to the `set` and `get` methods.

Unit Testing Framework: Generate Microsoft Word or Adobe PDF reports of a test run

Create easily readable, navigable, and archivable reports of your test results using the `TestReportPlugin`. You can generate reports in `.docx` and `.pdf` formats. For more information, see the `matlab.unittest.plugins.TestReportPlugin` class.

Unit Testing Framework: Improve continuous integration workflows using TAP Version 13 protocol and YAML diagnostics with the TAPPlugin

Use the `TAPPlugin` to produce a Test Anything Protocol (TAP) stream of your test output. You can integrate this stream into continuous integration systems like Jenkins™ or TeamCity®. The new `producingVersion13` method formats the stream using version 13 of the TAP format. With this format, you can include test diagnostics in YAML blocks. For more information, see the `producingVersion13` method of the `TapPlugin` class.

Unit and Performance Testing Frameworks: Improve code reuse and readability by using local functions in script-based tests

As of R2016b, MATLAB scripts can contain local functions. You can use local functions in your script-based unit tests to promote code reuse and readability by defining helper functions for your tests. For

an example of local, helper functions in a script-based test, see Write Script-Based Test Using Local Functions.

Unit and Performance Testing Frameworks: Tests that qualify that an empty character vector is a substring now pass

If you use the `verifyThat`, `assertThat`, `assumeThat`, or `fatalAssertThat` qualifications with the `IsSubstringOf` constraint to test that an empty character vector is a substring of any nonempty character vector, the qualification now passes. In previous versions of MATLAB, this qualification failed. For example, the following test now passes in MATLAB R2016b.

```
import matlab.unittest.constraints.IsSubstringOf

tc = matlab.unittest.TestCase.forInteractiveUse;
tc.verifyThat('',IsSubstringOf('any character vector'))
```

Compatibility Considerations

Prior to MATLAB release R2016b, tests such as `testCase.verifyThat('',IsSubstringOf('any character vector'))` failed. Update any tests that rely on the previous behavior.

Object Constructors: Construct objects faster with certain constraints

Significant overhead has been removed from object construction. These optimizations result in some changes in the behavior of object constructors, which are described in the next five items.

Compatibility Considerations

Class Destructor Called in Additional Conditions

In previous releases, when an error occurred in the constructor, MATLAB called the class destructor only if an object property was assigned a value in the constructor. With release R2016b, MATLAB also calls the class destructor when an error occurs under these conditions:

- A reference to the object is present in the code prior to the error.
- An early `return` statement is present in the code before the error.

These changes mean that MATLAB calls class destructors in more cases than in previous releases. The destructor must be able to operate on partially constructed objects. For more information, see Support Destruction of Partially Constructed Objects.

Returning Before Calling Superclass Not Allowed

In previous releases, constructors could return conditionally before calling the superclass constructor. With release R2016b, constructors cannot use early returns to conditionalize calls to the superclass constructor.

Revise any constructors that use returns to conditionalize calls to superclass constructors. For more information, see Subclass Constructors.

Path Change in Constructor Applies Immediately

In previous releases, path changes that remove the class of the executing constructor from the path did not affect the visibility of the class. With release R2016b, changes to the path made by the constructor are applied immediately. This new behavior is the same as that of path changes made in ordinary methods.

If a constructor removes the class folder from the path, MATLAB cannot access the class definition. The constructor can use the `mlock` function to keep the class definition in memory if changing the path in the constructor is necessary.

Changes to Error IDs

Error IDs have changed for certain constructor-related errors. These changes include the IDs for errors thrown for these conditions:

- Too many inputs
- Too many outputs
- Too few inputs
- Too few outputs
- Illegal superclass constructor call

Update any error handlers that rely on specific error IDs.

InstanceCreated Event Not Triggered When Constructor Does Not Return Object

If an object constructor does not return the constructed object, MATLAB does not trigger the `InstanceCreated` event. For more information, see Output Object Not Assigned.

You cannot listen for the `InstanceCreated` event if an object constructor does not return the constructed object to the caller.

Java: Pass 0-length arrays

MATLAB supports passing empty (0-length) Java arrays to Java methods that take an array of type other than `Object`.

Compatibility Considerations

Before MATLAB release R2016b, MATLAB converts zero-length Java arrays to null when the method takes an array of type other than `Object`.

Python Version 3.5: MATLAB support

MATLAB supports the following versions of CPython:

- Version 2.7
- Version 3.3
- Version 3.4
- Version 3.5

For more information, see Install Supported Python Implementation.

Python Version 3.3: Support to be phased out

Support for Python version 3.3 will be discontinued in a future release.

Compatibility Considerations

To ensure continued support for your applications, consider upgrading to another supported version of Python — version 3.4 or 3.5.

Source Control Integration: Customize external source control tools to use MATLAB to compare and merge

Use the MATLAB Comparison tool to review changes to files such as live scripts and MAT-files from your external source control client. For more information, see Customize External Source Control to Use MATLAB for Diff and Merge.

Source Control Integration: Include Git Submodules

To reuse code from another repository, you can specify Git submodules. For more information, see Add Git Submodules.

Source Control Integration: Fix problems with working copy locks using SVN Cleanup option.

Remove stale working copy locks using the **SVN Cleanup** option. For more information, see Get SVN File Locks.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	GNU gcc and gfortran version 4.9 Support for GNU gcc and gfortran version 4.7 is discontinued.	Linux
Added	Intel Parallel Studio XE 2016 for Fortran Intel Parallel Studio XE 2015 for Fortran	Mac OS X
Discontinued	Microsoft Visual Studio 2010 Professional	Windows
To be phased out	Visual C++ 2012 Professional Apple Xcode 6.2	Windows Mac OS X

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`matlab.mixin.SetGet` matches inexact property names using `set` and `get` methods.	Still runs	To enforce exact property name match, derive from `matlab.mixin.SetGetExactNames`.	Subclasses of `matlab.mixin.SetGet` allow inexact property name match by default.
Generating test suites from a class that derives from a concrete base class that defines methods which reference a `TestParameter`, `MethodSetupParameter`, or `ClassSetupParameter` not defined within the base class	Still runs	Define either the: • Base class as Abstract using the class-level Abstract attribute. For example, `classdef (Abstract) MyTest < matlab.unittest.TestCase`. • Abstract parameter properties for all parameters used by methods of the class. For example, `properties (Abstract,TestParameter)`.	In future releases, if your test class inherits from a concrete base class that uses a parameter that is not defined in the base class, MATLAB will throw an error.
Calling `import` from a script	Errors	Replicate the `import` statements in each function or script where you want to affect the import list.	The `import` function only affects the import list of the function or script within which it is used. The behavior of calling `import` at the command prompt is unchanged—it uses the import list for the command environment.
Calling scripts from class methods	Errors	Local functions defined in the class file have access to class members and might provide an alternate approach.	Class methods that call scripts that access private or protected class members from the script cause illegal access errors.
`'IgnoringCase'` option of these `matlab.unittest.constraints` classes: `CellComparator`, `PublicPropertyComparator`, and `StructComparator`	Errors	`'IgnoringCase'` option of the `IsEqualTo` constraint	Replace all instances of `'IgnoringCase'` in the `CellComparator`, `PublicPropertyComparator`, or `StructComparator` constraint with `'IgnoringCase'`in the `IsEqualTo` constraint.

Functionality	Result	Use This Instead	Compatibility Considerations
`'IgnoringWhitespace'` option of these `matlab.unittest.constraints` classes: `CellComparator`, `PublicPropertyComparator`, and `StructComparator`	Errors	`'IgnoringWhitespace'` option of the `IsEqualTo` constraint	Replace all instances of `'IgnoringWhitespace'` in the `CellComparator`, `PublicPropertyComparator`, or `StructComparator` constraint with `'IgnoringWhitespace'` in the `IsEqualTo` constraint.
`'Within'` option of these `matlab.unittest.constraints` classes: `CellComparator`, `PublicPropertyComparator`, and `StructComparator`	Errors	`'Within'` option of the `IsEqualTo` constraint	Replace all instances of `'Within'` in the `CellComparator`, `PublicPropertyComparator`, or `StructComparator` constraint with `'Within'` in the `IsEqualTo` constraint.

R2016a

Desktop

Live Editor: Create and run live scripts with embedded output; add equations and images to enhance the interactive narrative

Live scripts combine code, output, and formatted content together in a single interactive environment called the Live Editor. Add formatted text, images, hyperlinks, and equations to create an interactive narrative that can be shared with others.

In the Live Editor you can:

- Write, execute, and test code in a single interactive environment
- Generate results and graphics alongside the code that produced them.
- Run blocks of code individually or run the whole file.
- See errors at the location in the file where they occur.
- Include mathematical equations, images, and hyperlinks as supporting material.
- Share live scripts with other MATLAB users or convert them to `HTML` or `PDF` for publication.

For more information about live scripts, including incompatibilities and information about the Live Script file format, see What Is a Live Script?.

Toolboxes: Programmatically package and install custom MATLAB toolboxes with matlab.addons.toolbox package

Install, uninstall, query, and package toolboxes programmatically.

- To package toolboxes, use the `matlab.addons.toolbox.packageToolbox` function.
- To query or modify the version of a toolbox, use the `matlab.addons.toolbox.toolboxVersion` function.
- To install a toolbox, use the `matlab.addons.toolbox.installToolbox` function.
- To uninstall a toolbox, use the `matlab.addons.toolbox.uninstallToolbox` function.
- For information about installed toolboxes, use the `matlab.addons.toolbox.installedToolboxes` function.

Tab Completion: Complete parameter names and options in select MATLAB function calls

Complete parameter names and options when entering commands in MATLAB. For example, if you type `plot(1:10,'` and press the **Tab** key, MATLAB displays a list of possible parameter names and options for the `plot` function.

Completion of parameters and options is not available for all functions.

Pause Button: Pause the execution of a program from the Editor and enter debug mode

To pause the execution of a program while it is running, in the **Editor** tab, click the **Pause** button. MATLAB pauses execution at the next executable line.

Toolboxes: Customize additions to MATLAB path upon toolbox installation

You can indicate which folders are excluded from the user's MATLAB path when they install your toolbox. By default, any of the toolbox folders that are on your path when you create the toolbox are added to the user's path. For more information, see Create and Share Toolboxes.

Preferences: Migrate preferences from MATLAB versions up to three releases preceding the release starting up

MATLAB can now migrate preferences from MATLAB versions up to three releases immediately preceding the release starting up. For example, if you start MATLAB `R2016a` for the first time and a preferences folder exists for `R2014b`, MATLAB migrates the preferences from `R2014b` to `R2016a`. If two or more preferences folders for the previous three releases exist, MATLAB imports the preferences from the *latest* preferences folder. For more information, see Preferences MATLAB Uses When Multiple Releases Are Installed.

Compatibility Considerations

On Mac platforms, migrating preferences might result in an out-of-date MEX options file. For more information, see SDK "macosx10.8" cannot be located Error Message.

Internationalization: Default encoding scheme on Mac platforms will change in a future release

Currently, the default encoding scheme on Mac platforms is determined by the OS X user locale setting. In a future release, the default encoding scheme will be UTF-8.

Compatibility Considerations

In a future release, the UTF-8 version of the MathWorks locale database will be the default on Mac OS X systems. Any text file created in MATLAB that is encoded in the user default encoding and contains characters other than 7-bit ASCII characters must be converted to the UTF-8 encoding scheme. Otherwise, MATLAB and other MathWorks products might not be able to load the files properly.

Do not convert text files to UTF-8 if you share them with users on Windows platforms.

How to Change the MATLAB Default Encoding to UTF-8

If you have text files containing non-7-bit ASCII characters, you must convert the encoding in the files before changing the default encoding on your computer. For instructions, see "How to Convert Text File Encoding to UTF-8" on page 18-4.

You can change the default encoding scheme on Mac OS X platforms by using the UTF-8 locale database found in the *`matlabroot`*`/bin` folder.

To change the default locale database, type:

```
mldir = fullfile(matlabroot,'bin');
copyfile(fullfile(mldir,'lcdata.xml'),...
  fullfile(mldir,'lcdata_default.xml'));
copyfile(fullfile(mldir,'lcdata_utf8.xml'),...
  fullfile(mldir,'lcdata.xml'));
```

Alternatively, you can change the default locale database by setting an environment variable on your computer:

1. Set the environment variable, `MWLOCALE_LCDATA_FILENAME`, with the UTF-8 version of the locale database name, `lcdata_utf8.xml`, in the environment file, `environment.plist`.
2. Close MATLAB if it is currently running.
3. Start MATLAB by double-clicking the MATLAB icon in the Applications folder.

How to Convert Text File Encoding to UTF-8

Before converting a file, copy the file to a new directory.

Use the Mac OS X TextEdit application to convert the file encoding. For example:

1. Open a MATLAB text file with TextEdit.
2. Select **File** -> **Save as...**
3. Change the file name.
4. Change **Plain Text Encoding:** to `Unicode (UTF-8)`.
5. Save the file.

Alternatively, the following MATLAB function, `convert_file_encoding`, creates a new text file with different encoding.

```
function convert_file_encoding(infile,outfile,from_encoding,to_encoding)

if strcmp(infile(end-2:end),'mdl');
    isMDL = 1;
else
    isMDL = 0;
end


fpIn = fopen(infile,'r','n',from_encoding);
fpOut = fopen(outfile,'w','n',to_encoding);

while feof(fpIn) == 0
    lineIn = fgets(fpIn);
    if isMDL && strncmp('SavedCharacterEncoding',strtrim(lineIn),22)
        lineIn = regexprep(lineIn,from_encoding,to_encoding);
    end

    fwrite(fpOut,lineIn,'char');
end
```

```
fclose(fpIn);
fclose(fpOut);
end
```

To use this function, you need to know the current encoding, `from_encoding`.

- For MATLAB R2010b or later:

  ```
  ret = feature('locale');
  from_encoding = ret.encoding;
  ```

- For MATLAB R2008a through R2010a:

  ```
  ret = feature('locale');
  [t,r] = strtok(ret.ctype,'.');
  from_encoding = r(2:end);
  ```

For example, a file, `myFile.m`, was created with MATLAB encoding set to `ISO-8859-1`. To convert the file to `UTF-8`, type:

```
convert_file_encoding('myFile.m','myFileUTF8.m','ISO-8859-1','UTF-8')
```

Language and Programming

datetime Object: Set the default locale and format of datetime objects through the Preference panel

View and modify the default locale and format of `datetime` objects in the Command Window Preferences. For more information, see Set Command Window Preferences.

zeros, ones, and eye Functions: Create logical arrays

You can specify `logical` output with the `zeros`, `ones`, and `eye` functions.

cellstr, deblank, and strtrim Functions: Keep significant whitespace characters when removing leading or trailing whitespace

The `cellstr`, `deblank`, and `strtrim` functions no longer remove significant leading or trailing whitespace characters, such as nonbreaking space characters. The most common whitespace characters that are significant are `char(133)`, `char(160)`, `char(8199)`, and `char(8239)`.

Compatibility Considerations

Prior to MATLAB release R2016a, the `cellstr`, `deblank`, and `strtrim` functions removed all leading or trailing whitespace characters, including significant whitespace characters.

rowfun and varfun Functions: Create output table without row names when using the 'GroupingVariables' parameter

The `rowfun` and `varfun` functions no longer add row names to the output table when you use the `'GroupingVariables'` parameter to specify the grouping variables.

Compatibility Considerations

Prior to MATLAB release R2016a, the `rowfun` and `varfun` functions added row names to the output table when using the `'GroupingVariables'` parameter. To add row names, assign them to the `RowNames` property of the output table. For more information on table properties, see `Table Properties`.

Debugging: Set breakpoints while MATLAB is executing

You can now set breakpoints at any time, whether MATLAB is idle or busy running a file.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Calling `nargin`, `nargout`, or `inputname` from within a script	Still runs	Move calls to `nargin`, `nargout`, and `inputname` into the relevant function file. If you need the results in a script, call the functions and store the results in a variable. Then use the variable in the script.	If you call `nargin`, `nargout`, or `inputname` from within a script, MATLAB throws an error in future releases.
Visibility of local functions	Still runs	Store the function handle to the local function in a variable and use that instead of the call to the local function. For example, assign `fcn = @myLocalFcn` and update the script to use the variable `fcn` instead of calling `myLocalFcn` directly.	In future releases, local functions are visible only to other functions in the same file. Currently, if a function file that defines local functions invokes a script, that script can call those local functions.
Visibility of private functions	Still runs	Moving scripts that need access to private functions to the folder immediately above the `private` subfolder might provide an alternate approach.	In future releases, private functions are visible only to other functions or scripts in the folder immediately above the `private` subfolder. Currently, private functions are available to scripts outside the parent folder that are called by functions residing in the parent folder.
`nargchk`	Warns	`narginchk`	Replace all instances of `nargchk` with `narginchk`.

Mathematics

Moving Statistics Functions: Calculate moving statistics using the movmean, movsum, movmedian, movmax, movmin, movvar, and movstd functions

You now can compute statistics in a sliding window along a dimension of an array with the `movmean`, `movsum`, `movmedian`, `movmax`, `movmin`, `movvar`, and `movstd` functions.

datetime and duration Arrays: Compute standard deviations with std

The `std` function can calculate the standard deviation of a `datetime` or `duration` array.

datetime and duration Arrays: Ignore NaNs and NaTs using 'omitnan' or 'omitnat' in the functions mean, median, std, and sum

You now can exclude NaNs from `duration` arrays when using the `mean`, `median`, `std`, and `sum` functions.

You also can exclude NaTs from `datetime` arrays when using the `mean`, `median`, and `std` functions.

graph and digraph Classes: Analyze graphs and networks using centrality and nearest nodes functions

The `centrality` function computes several different types of node centrality, such as `'degree'`, `'betweenness'`, and `'pagerank'`. The `nearest` function calculates the nearest nodes within a specified distance of a source node.

svds Function: Compute singular values with improved performance and convergence behavior with a wide variety of matrices

The algorithm of the `svds` function shows improved performance and convergence behavior with a wide variety of matrices.

Compatibility Considerations

Three new options for `sigma` are being introduced to `svds` to calculate the largest and smallest singular values of a matrix: `'largest'`, `'smallest'`, and `'smallestnz'`. Previously, using a value of `0` for `sigma` was common when calculating the smallest singular values.

`svds(A,k,0)` now returns a warning. Use `svds(A,k,'smallest')` to find the smallest singular values instead.

median Function: Compute medians with improved performance

The `median` function computes the median of an array faster.

cummin, cummax, cumprod, and cumsum Functions: Compute cumulative minimum, maximum, product, and sum with improved performance

Calculating the cumulative minimum, maximum, product, and sum with the `cummin`, `cummax`, `cumprod`, and `cumsum` functions is now faster.

GraphPlot Objects: Interactively inspect graph plots using data cursor and plot selection

You now can use the data cursor to select nodes in a graph and display basic information, such as the node ID and degree. See Display Data Values Interactively for more information about data cursors, or Add Node Properties to Graph Plot Data Cursor for an example involving a graph plot.

Selecting a graph plot also enables the use of plot tools to interact with the graph, such as Plot Browser and Property Editor. See Customize Graph Using Plot Tools for more information.

Graphics

polarplot Function: Plot data in polar coordinates and modify properties of polar axes

To plot data in polar coordinates, use the `polarplot` function. You can modify the resulting polar axes, such as changing the limits or orientation, by setting Polar Axes Properties.

The `polarplot` function supersedes the `polar` function.

yyaxis Function: Create charts with two y-axes and customize each y-axis individually

To create charts with two *y*-axes, use the `yyaxis` function. You can plot data against either *y*-axis using common charting functions, such as `plot` or `errorbar`. You can customize each *y*-axis by setting properties of the axes object.

The `yyaxis` function supersedes the `plotyy` function. For an example of its use, see Create Chart with Two y-Axes.

Legend Object: Add legend title and create callbacks to highlight plots when clicking legend items

To add a title to a legend, use the new `Title` property of the legend or using the `title` function, for example:

```
plot(rand(3))
l = legend('Line 1','Line 2','Line 3');
title(l,'My Legend')
```

Also, now you can highlight charts in the axes when you click the associated legend item. Set the new `ItemHitFcn` property of the legend to a callback function that changes properties of the charts. For an example, see Create Interactive Legends Using Callbacks.

histogram2 Function: Enable data linking and brushing for bivariate histograms

You now can link and brush data in `histogram2` plots.

Function Plots: Visualize mathematical expressions as parametric line, surface, and contour plots

Plot mathematical expressions using a family of new and updated function plots. These functions supersede the existing `ez` family of functions, such as `ezplot`.

- `fplot` plots 2-D lines, including parametric lines. Supersedes `ezplot`.
- `fplot3` plots 3-D parametric curves. Supersedes `ezplot3`.
- `fcontour` plots 2-D contours. Supersedes `ezcontour`.

- `fsurf` plots 3-D surfaces, including parametric surfaces. Supersedes `ezsurf`.
- `fmesh` plots 3-D meshes, including parametric meshes. Supersedes `ezmesh`.

Compatibility Considerations

The behavior of `fplot` changed in these ways:

- Issues a warning when using a string to define the function to plot. Instead, use an anonymous function or a function handle.
- Issues a warning when returning two output arguments. Instead, return the function line object and use its `XData` and `YData` properties.
- No longer supports specifying the *y*-axis limits as an input argument. Instead, use the `ylim` function.
- No longer supports specifying the number of evaluation points as an input argument. Instead, specify the `MeshDensity` property.
- No longer supports specifying the error tolerance as an input argument.

Graphics Display: Render plots with large numbers of markers faster

Graphics that contain markers have improved performance and quality. For more information, see `opengl`.

3-D Pan and Zoom: Explore data with improved pan and zoom behavior for axes in a 3-D view

For axes in a 3-D view, panning and zooming now shift the view of the data by modifying the axis limits, instead of moving the entire axes. The axes box stays in the same location within the figure. Previously, panning and zooming modified camera properties, which moved the entire axes around within the figure.

Compatibility Considerations

If you have an axes in a 3-D view, then the behavior of pan and zoom is different. For the old behavior, use one of these options:

- Use the context menus when in pan or zoom mode. For example, when you are in zoom mode, right-click over the axes and select **3D Options > Camera Pan and Zoom**.
- Use the `setAxes3DPanAndZoomStyle` function to specify the behavior, for example:

  ```
  ax = gca;
  z = zoom;
  setAxes3DPanAndZoomStyle(z,ax,'camera')
  ```

- Use the camera toolbar. To view the camera toolbar, select **View > Camera Toolbar** from the figure menu.

Graphics Drivers: Use latest drivers to avoid instabilities with older NVIDIA Windows drivers

If you are using a Windows NVIDIA graphics driver older than version 9.18.13.4052 (or 340.52 in NVIDIA driver version convention), then MATLAB reverts to using software OpenGL® instead of

hardware-accelerated OpenGL. Hardware-accelerated OpenGL is not available by default due to known instabilities with the driver when running MATLAB. To take advantage of all the visual and performance benefits of hardware-accelerated OpenGL, update your graphics driver to the latest version. For more information, see System Requirements for Graphics.

Printed Figure Size: Print or save figures that match size of the figure on the screen by default

Printed and saved figures now match the size of the figure on the screen by default. Previously, printed and saved figures were 8-by-6 inches by default. This change affects figures printed to a printer or saved to a bitmap image or vector graphics file format, for example, using the `print` and `saveas` functions.

Compatibility Considerations

The size of printed or saved figures might be different because now the default value for the `PaperPositionMode` property of figures is `'auto'` instead of `'manual'`. To get the previous behavior, set the default value back to `'manual'` using one of these techniques:

- Use the new print preference option. Print preferences persist across MATLAB sessions. You can set the print preference to either `'auto'` (new behavior) or `'manual'` (previous behavior), for example:

  ```
  matlab.graphics.internal.setPrintPreferences('DefaultPaperPositionMode','manual')
  ```

  To query the current print preference value, use the following command. If you set a preference, then the command returns `'auto'` or `'manual'`. If you did not set a preference, then the command returns `'unset'`.

  ```
  matlab.graphics.internal.getPrintPreferences
  ```

- Set the default value on the root object. This option affects only new figures in the current MATLAB session.

  ```
  set(groot,'defaultFigurePaperPositionMode','manual')
  ```

  For an existing figure, set the `PaperPosition` property to `'default'`, for example:

  ```
  set(gcf,'PaperPosition','default')
  ```

print Function: Print figures that fill page using the '-fillpage' and '-bestfit' options

Print or save figures that fill the page using the new `'-fillpage'` and `'-bestfit'` options for the `print` function. These options maximize the size of the figure while leaving a minimum page margin of 0.25 inch. The `'-bestfit'` option preserves the figure's aspect ratio. The `'-fillpage'` option ignores the aspect ratio. Both options are valid only when printing a figure to a printer or saving it to a paged format (PDF and full page PostScript®).

Figure Menu: Save figures that honor the PaperPosition value using File > Save As

When you save a figure using **File > Save As**, the size of the saved figure now honors the figure's `PaperPosition` property value. By default, the `PaperPosition` property recalculates according to

the size of the figure on the screen. As a result, the saved figure matches the size of the figure on the screen. However, if you change the `PaperPosition` property value, then the saved figure uses the specified size. Previously, the saved figure matched the size of the figure on the screen regardless of the `PaperPosition` property value. For the old behavior, set the figure's `PaperPositionMode` property to `'auto'` before saving the figure:

```
set(gcf,'PaperPositionMode','auto')
```

Compatibility Considerations

The size of saved figures might be different.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`plotyy` function	Still runs	`yyaxis`	Replace all instances of `plotyy` with `yyaxis`.
`polar` function	Still runs	`polarplot`	Replace all instances of `polar` with `polarplot`.
`ezplot` function	Still runs	`fplot`	Replace all instances of `ezplot` with `fplot`.
`ezplot3` function	Still runs	`fplot3`	Replace all instances of `ezplot3` with `fplot3`.
`ezcontour` function	Still runs	`fcontour`	Replace all instances of `ezcontour` with `fcontour`.
`ezcontourf` function	Still runs	`fcontour`	Replace all instances of `ezcontourf(...)` with `fcontour(...,'Fill','on')`.
`ezsurf` function	Still runs	`fsurf`	Replace all instances of `ezsuf` with `fsurf`.
`ezsurfc` function	Still runs	`fsurf`	Replace all instances of `ezsurfc(...)` with `fsurf(...,'ShowContours', 'on')`.
`ezmesh` function	Still runs	`fmesh`	Replace all instances of `ezmesh` with `fmesh`.
`ezmeshc` function	Still runs	`fmesh`	Replace all instances of `ezmeshc(...)` with `fmesh(...,'ShowContours', 'on')`.
`PaperPositionMode` property of figure objects	Default value now `'auto'`	Not applicable	Printed and saved figures now match the size of the figure on the screen by default.

Functionality	Result	Use This Instead	Compatibility Considerations
`[l,icons,plots,txt] = legend(__)` syntax	Still runs	`l = legend(__)` syntax	Remove all instances of the `[l,icons,plots,txt] = legend(__)` syntax. You can make some modifications to the legend by returning the legend object and setting legend properties instead. For a list, see Legend Properties. For information on new legend properties, see “Legend Object: Add legend title and create callbacks to highlight plots when clicking legend items” on page 9-10.

App Building

App Designer: Build MATLAB apps with line and scatter plots using an enhanced design environment and an expanded UI component set

App Designer is a rich application development environment for developing apps. Major App Designer features include:

- Numerous user interface components

  In addition to standard components, such as buttons, check boxes, and panels, App Designer also offers gauges, lamps, knobs, and switches.
- Integrated editor, property sheets, and property inspector

  The editor generates code as you lay out your app and customize component properties. The editor has coding alerts and many of the debugging features that are available in the MATLAB Editor. The integrated property sheets enable you to set commonly used properties for components, such as default component values, text, and text font styles. The property inspector provides access to all writable properties.
- Enhanced code structure

  App Designer generates code based on your component selection and layout. App Designer structures the code for readability and does not generate unnecessary code. In addition, Areas that App Designer manages are set to read-only access, so you cannot accidentally overwrite or delete required code.

If you built apps using GUIDE in the past, be aware that there are some important differences between GUIDE and App Designer in terms of functionality and support for graphics workflows. For more information, see:

- Differences Between App Designer and GUIDE
- Graphics Support in App Designer

To get started using App Designer, see Create Simple App Using App Designer.

Data Import and Export

writetable Function: Write to text files significantly faster, especially for large files

Performance improvement of the `writetable` function increases with table size.

readtable Function: Read from Excel files with faster performance

The `readtable` function reads faster from Excel files.

writetable Function: Write to Excel files on Mac and Linux platforms

In addition to Windows, the `writetable` function now writes to Excel files on Mac and Linux.

Compatibility Considerations

If you specify a range that exceeds the data being written, then `writetable` will not write to those cells. Previously, `writetable` filled the remaining cells with `#N/A`.

When writing `NaT` `datetime` values or `<undefined>` `categorical` values, `writetable` writes empty cells instead of `NaT` and `<undefined>`, respectively.

spreadsheetDatastore Function: Import and process data from a collection of Excel files

The `spreadsheetDatastore` function creates a `SpreadSheetDatastore` object for processing large collections of Excel files.

datastore Function: Import a TabularTextDatastore object with improved file format detection

The `datastore` function now detects file formatting more accurately when creating a `TabularTextDatastore` object.

The `datastore` function now automatically includes only files with extensions `.txt`, `.csv`, `.dat`, `.dlm`, `.asc`, and `.text` by default.

Compatibility Considerations

If you want to include unsupported text file extensions, then specify them using the `'FileExtensions'` parameter in the `datastore` function.

ImageDatastore Object: Specify image labels using the Labels property and process with splitEachLabel, countEachLabel, and shuffle functions

You now can organize an `ImageDatastore` according to specified labels using the `Labels` property. When creating an `ImageDatastore` object, you can use the `LabelSource` parameter to assign label names according to the file folder names. Labeling is useful for workflows in machine learning and neural networks, for example.

The `splitEachLabel` function creates new `ImageDatastore` objects according to a specified number or percentage of images from each label. `splitEachLabel` can also draw images randomly. This function is useful for testing and training sets, for example.

The `countEachLabel` function creates a summary table of all labels in the `ImageDatastore` and how many files belong to each.

The `shuffle` function randomly reorders the files in the original `ImageDatastore` and creates a new one.

fileDatastore Function: Create a custom datastore for a file collection too large to fit in memory

The `fileDatastore` function creates a custom `FileDatastore` object that uses a specified custom read function to import the files.

readtable Function: Read text files with automatic detection of delimiters, header lines, and variable names

The `readtable` function automatically detects delimiters, header lines, and variable names in text files.

Compatibility Considerations

You can restore the previous `readtable` defaults by specifying the delimiter, header line, or variable name parameter values explicitly, for example,

```
readtable('file.txt','Delimiter',',','HeaderLines',0,'ReadVariableNames',true
);
```

tabularTextDatastore and imageDatastore Functions: Create objects to import large text and image data collections

You now can use the `tabularTextDatastore` function to conveniently create a `TabularTextDatastore` object for large collections of text files.

You can use the `imageDatastore` function to conveniently create an `ImageDatastore` object for large collections of image data.

writetable Function: Detect text with embedded delimiters automatically and write as quoted text

The `writetable` function automatically detects and adds quotations around text, `categorical`, and `datetime` variables containing an embedded delimiter (such as `'\t'`).

TabularTextDatastore Objects: Read text files with automatic detection of delimiters, header lines, and variable names

You now can read from a `TabularTextDatastore` object with automatic detection of the `Delimiter`, `NumHeaderLines`, and `ReadVariableNames` properties. For more information, see TabularTextDatastore Properties.

Compatibility Considerations

You can restore the previous `TabularTextDatastore` defaults by specifying the delimiter, header line, or variable name parameter values explicitly, for example, `tabularTextDatastore('file.txt','Delimiter',',','NumHeaderLines',0,'ReadVariableNames',true);`

imread Function: Generate C-code using MATLAB Coder

The `imread` function generates C-code for 8-bit grayscale and 24-bit truecolor JPEG images using a platform-specific shared library.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`KeyValueLimit` property of the `KeyValueDatastore` class	Errors	`ReadSize` property of `KeyValueDatastore` class	Replace all instances of `KeyValueLimit` with `ReadSize`.
`RowsPerRead` property of the `TabularTextDatastore` class	Errors	`ReadSize` property of `TabularTextDatastore` class	Replace all instances of `RowsPerRead` with `ReadSize`.
Folders and wildcard (*) characters in the `Files` property of the `KeyValueDatastore` and `TabularTextDatastore` classes	Errors	Full paths and file names	Replace all folder names or wildcard characters with full path and file names when assigning to the `Files` property.

Performance

Performance Testing Framework: Measure MATLAB code performance using the unit testing framework

Measure the performance of your MATLAB code using the performance testing framework. The framework includes performance measurement-oriented features such as running your code several times to "warm-up" the code and accounting for noise in the measurements.

The performance test interface leverages the script, function, and class-based unit testing interfaces. Therefore, you can perform qualifications within your performance tests to ensure correct functional behavior while measuring code performance. Also, you can run your performance tests as standard regression tests to ensure that code changes do not break performance tests.

For more information, see Performance Testing Framework.

Graphics Display: Render plots with large numbers of markers faster

Graphics that contain markers have improved performance and quality. For more information, see `opengl`.

writetable Function: Write to text files significantly faster, especially for large files

Performance improvement of the `writetable` function increases with table size.

readtable Function: Read from Excel files with faster performance

The `readtable` function reads faster from Excel files.

median Function: Compute medians with improved performance

The `median` function computes the median of an array faster.

cummin, cummax, cumprod, and cumsum Functions: Compute cumulative minimum, maximum, product, and sum with improved performance

Calculating the cumulative minimum, maximum, product, and sum with the `cummin`, `cummax`, `cumprod`, and `cumsum` functions is now faster.

Hardware Support

Raspberry Pi: Acquire images from USB webcams

Use the MATLAB Support Package for Raspberry Pi Hardware to bring live images into MATLAB from USB webcams attached to your Raspberry Pi hardware board.

With simple MATLAB functions preview and acquire single snapshots from the camera, and optionally set up a loop of acquired images. The `webcam` function creates the USB Camera object used to acquire images, and the `snapshot` function returns a single image from the camera.

Arduino: Build custom add-ons to interface with additional hardware and software libraries

MATLAB Support Package for Arduino Hardware provides a framework of classes that you can use to create custom applications to use with your Arduino device. The custom applications can be code, libraries, or devices that work with an Arduino. For more information, see Custom Arduino Add-On Device, Library, and CodePerformance Testing Framework in the MATLAB Support Package for Arduino Hardware documentation.

To install the support package, on the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons > Get Hardware Support Packages**.

Advanced Software Development

Performance Testing Framework: Measure MATLAB code performance using the unit testing framework

Measure the performance of your MATLAB code using the performance testing framework. The framework includes performance measurement-oriented features such as running your code several times to "warm-up" the code and accounting for noise in the measurements.

The performance test interface leverages the script, function, and class-based unit testing interfaces. Therefore, you can perform qualifications within your performance tests to ensure correct functional behavior while measuring code performance. Also, you can run your performance tests as standard regression tests to ensure that code changes do not break performance tests.

For more information, see Performance Testing Framework.

Unit Testing Framework: Quickly create explicit test suites using testsuite function

Use the `testsuite` function to create a suite of tests from your current working folder or from specified test content. If you have an explicit test suite, you can use test selectors and test runners.

In previous releases of MATLAB, you create explicit test suites with the methods in the `matlab.unittest.TestSuite` class. The `testsuite` function provides a more convenient means to create test suites.

Unit Testing Framework: Access diagnostic information recorded on test results

You can programmatically access the diagnostic information on test results using a `DiagnosticsRecordingPlugin`. The `DiagnosticsRecordingPlugin` class records test qualification failures and events logged at `Terse` verbosity. However, you can configure the plugin to record passing diagnostics and logged messages at different verbosity levels.

If you run tests with the `runtests` function or the `run` method of `TestSuite` or `TestCase`, then the test framework uses a `DiagnosticsRecordingPlugin` plugin by default. Also, if you run performance tests with the `runperf` function or the `run` method of `TimeExperiment`, then the test framework uses this plugin by default.

For more information, see `matlab.unittest.plugins.DiagnosticsRecordingPlugin`.

Unit Testing Framework: Create temporary working folder using the WorkingFolderFixture

The `matlab.unittest.fixtures.WorkingFolderFixture` class provides a fixture that creates a temporary folder and sets it as the current working folder. The test or the product under test can create files and modify the contents of the folder without affecting the source code or test folder structure.

Although both the `WorkingFolderFixture` and `TemporaryFolderFixture` classes create a temporary folder, the `WorkingFolderFixture` also sets the folder as the current working folder. At fixture teardown, the testing framework restores the current working folder to its previous state.

For more information, see `matlab.unittest.fixtures.WorkingFolderFixture`.

Unit Testing Framework: Test set membership and uniqueness with HasUniqueElements, IsSubsetOf, and IsSupersetOf constraints

You can test set membership and uniqueness of the actual value set using set constraints in the `matlab.unittest.constraints` package. For example:

- `HasUniqueElements` tests if an actual value set has unique elements.
- `IsSubsetOf` tests if an actual value set is a subset of the expected value set.
- `IsSupersetOf` tests if an actual value set is a superset of the expected value set.

Unit Testing Framework: Set up custom fixture to delegate work to another fixture

The `matlab.unittest.fixtures.Fixture` class enables you to create custom fixtures to configure the environment state required for tests. The `applyFixture` method of the `Fixture` class enables your custom fixture to delegate work to another fixture. When the testing framework tears down the custom fixture, it also tears down the fixture you used to set it up.

For more information, see `matlab.unittest.fixtures.Fixture.applyFixture`.

Unit Testing Framework: Exclude specified fields and properties from constraint comparison

When you compare structures using the `IsEqualTo` or `StructComparator` classes, specify which fields to ignore using the `'IgnoringFields'` name-value pair argument.

When you compare objects using the `PublicPropertyComparator` with the `supportingAllValues` static method, specify which properties to ignore using the `'IgnoringProperties'` name-value pair argument. If the values are structures, then specify which fields to ignore using the `'IgnoringFields'` option.

Unit Testing Framework: Customize how the PathFixture fixture adds folders to the path

Now,you can create a `PathFixture` fixture that includes subfolders of the specified folder on the path. For more information, see the `'IncludingSubfolders'` name-value pair argument of `PathFixture`.

Additionally, you can have the `PathFixture` fixture add the folder to the end (bottom) of the path. By default, the fixture adds a folder to the beginning (top) of the MATLAB path. For more information, see the `'Position'` name-value pair argument of `PathFixture`.

Property Definition: Restrict the class of property values

Restrict property values to a specified class or a class compatible with the specified class. For more information, see Restrict Class of Properties.

Property Definition: Define only one property per line in class definitions

Define only one property per line in the property block of a class definition. Also, you cannot define properties on the same line as the `properties` or `end` keywords.

Compatibility Considerations

In releases R2016a, MATLAB interprets any name following a property name as a restricting class. The second name is not interpreted as a property name. More than two names on a line cause an error. A single line in a property block can contain the property name, an optional restricting class, and an optional default value. Default values must be of a class that is compatible with the class restriction.

```
properties
   PropName ClassName = defaultValue;
end
```

You must change any existing property blocks that list more than one property per line to list only one property per line. The following list describes cases that you must change:

- The `properties` keyword is on the same line as property names.
- The `end` keyword is on the same line as property names.
- The entire properties block is on one line using white-space separators.
- More than two property names are on one line.

MATLAB interprets two property names on one line as a restricted property. If the second name is not a valid class name, MATLAB generates an error.

event.hasListener Function: Determine if an event has listeners

The `event.hasListener` function returns a logical `true` or `false` indicating if listeners exist for a specific event name and event source.

event.DynamicPropertyEvent Class: Get dynamic property name from event data

The `PropertyAdded` and `PropertyRemoved` events provide an `event.DynamicPropertyEvent` object as the event data that is passed to listeners of those events. This event data object has a property, `PropertyName`, that contains the name of the dynamic property whose addition or removal triggered the respective event.

Enumerations: Substitute character arrays using new class methods

Enumerations have improved compatibility with character arrays and cell arrays of strings. This compatibility enables conversion of characters to enumerations and the use of enumeration members

in set membership and string comparison functions. For more information, see Conversion of Characters to Enumerations and Operations on Enumerations.

waitfor Function: Suspend execution of pending operations on any handle object

With release R2016a, the `waitfor` function works with any instance of a MATLAB handle class.

Source Control Integration: Display relation of local changes to remote branch

If you are using Git for source control, you can view information about your current branch relative to the remote tracking branch in the repository. Right-click the file or white space of the Current Folder browser and select **Source Control > View Details**. The **Git information** field indicates whether your committed local changes are ahead of, behind, or coincident with the remote tracking branch.

Source Control Integration: View summary status icon for folders

If you have folders in your source control repository, the Current Folder browser shows an icon for the folder that summarizes the status of the contents of the folder. The tooltip for the icon displays a summary of the changes in the folder.

If the source control system supports storing folders, the Current Folder browser displays the appropriate status icon for the folder and its contents. Otherwise, it displays either a dot indicating that the folder is not under source control or the Modified icon  to indicate that files within the folder are modified. SVN supports storing folders, but Git does not.

MATLAB Engine for Python: Start or connect asynchronously to MATLAB from Python

You can start the MATLAB Engine for Python asynchronously using the `async` keyword argument in the `matlab.engine.start_matlab` and `matlab.engine.connect_matlab` commands.

MATLAB builds with Boost library version 1.56.0

MATLAB builds with an updated version of the Boost library, version 1.56.0.

Compatibility Considerations

If you use any Boost library in a MEX function, you might not be able to load that function if you do not use Boost library version 1.56.0.

MEX command does not accept .bat or .sh compiler options files

The `mex` command `-f` option does not accept arguments with `.bat` or `.sh` file extensions.

Instead of using the `-f` option to specify a compiler, use the work flows described in Change Default Compiler.

You can modify compiler build options using `mex` command options. For example, use the `-L` and `-I` options to locate libraries on your system. Use the *`varname=varvalue`* option to pass options to the compiler.

For the complete list of `mex` command options, see the `option1 ... optionN` input argument.

Compiler support changed for building MEX files and standalone MATLAB engine and MAT-file applications

Support	Compiler	Platform
Added	Intel Parallel Studio XE 2016 Intel Parallel Studio XE 2015 Intel C++ Composer XE 2013 with Microsoft Visual Studio 2013	Windows
Added	Apple Xcode 7.x	Mac OS X
Discontinued	Microsoft Visual Studio 2008 Professional Edition Intel C++ Composer XE 2011 with Microsoft Visual Studio 2010 Intel Visual Fortran Composer XE 2011 with Microsoft Visual Studio 2010	Windows
To be phased out	Microsoft Visual Studio 2010 Professional Edition Microsoft Windows SDK 7.1	Windows

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
Calling `import` from a script	Still runs	Replicate the `import` statements in each function or script where you want to affect the import list.	In future releases, the `import` function only affects the import list of the function or script within which it is used. Currently, if you call `import` in a script that is invoked from a function, it affects the import list of the function. The behavior of calling `import` at the command prompt is unchanged—it uses the import list for the command environment.
Calling `clear import` from a script	Still runs	In future releases, most of the calls to `clear import` from a script are unnecessary because the `import` statements are scoped to the script.	In future releases, calling `clear import` from a function or script returns an error. Currently, this action returns an error from within a function.
Calling scripts from class methods	Still runs	Local functions defined in the class file have access to class members and might provide an alternate approach.	In future releases, class methods that call scripts that access private or protected class members from the script cause illegal access errors.
`'Recursively'` option of `runtests`	Still runs	`'IncludeSubfolders'` and `'IncludeSubpackages'` options of `runtests`	In calls to `runtests`, replace all instances of `'Recursively'` with `'IncludeSubfolders'` and/or `'IncludeSubpackages'`
Creating a dynamic property with an invalid name	Errors	For valid names, see Name Dynamic Properties.	Creating a dynamic property with an invalid name causes an `InvalidDynamicPropertyName` error. Previous releases issued a warning.

Functionality	Result	Use This Instead	Compatibility Considerations
`'IgnoringCase'` option of these `matlab.unittest.constraints` classes: `CellComparator`, `PublicPropertyComparator`, and `StructComparator`	Warns	`'IgnoringCase'` option of the `IsEqualTo` constraint	Replace all instances of `'IgnoringCase'` in the `CellComparator`, `PublicPropertyComparator`, or `StructComparator` constraint with `'IgnoringCase'`in the `IsEqualTo` constraint.
`'IgnoringWhitespace'` option of these `matlab.unittest.constraints` classes: `CellComparator`, `PublicPropertyComparator`, and `StructComparator`	Warns	`'IgnoringWhitespace'` option of the `IsEqualTo` constraint	Replace all instances of `'IgnoringWhitespace'` in the `CellComparator`, `PublicPropertyComparator`, or `StructComparator` constraint with `'IgnoringWhitespace'` in the `IsEqualTo` constraint.
`'Within'` option of these `matlab.unittest.constraints` classes: `CellComparator`, `PublicPropertyComparator`, and `StructComparator`	Warns	`'Within'` option of the `IsEqualTo` constraint	Replace all instances of `'Within'` in the `CellComparator`, `PublicPropertyComparator`, or `StructComparator` constraint with `'Within'` in the `IsEqualTo` constraint.

R2015aSP1

Version: 8.5.1

Bug Fixes

R2015b

Desktop

Add-On Explorer: Add capabilities to MATLAB, including community-authored and MathWorks toolboxes, apps, functions, models, and hardware support

Add-ons extend the capabilities of MATLAB by providing additional functionality for specific tasks and applications, such as connecting to hardware devices, additional algorithms, and interactive apps. Add-ons encompass a wide variety of resources, including products, apps, support packages and toolboxes, and are available from MathWorks and from the global MATLAB user community.

Using the new Add-On Explorer, you can browse and install available add-ons directly from MATLAB.

- To access the Add-On Explorer in MATLAB, go to the **Home** tab and click the **Add-Ons**  icon.
- To manage installed add-ons, go to the **Home** tab, and select **Add-Ons** > **Manage Add-Ons**.

For more information, see the Add-Ons documentation.

Compatibility Considerations

To continue using your apps and custom toolboxes from previous MATLAB installations, you must migrate them to the current release:

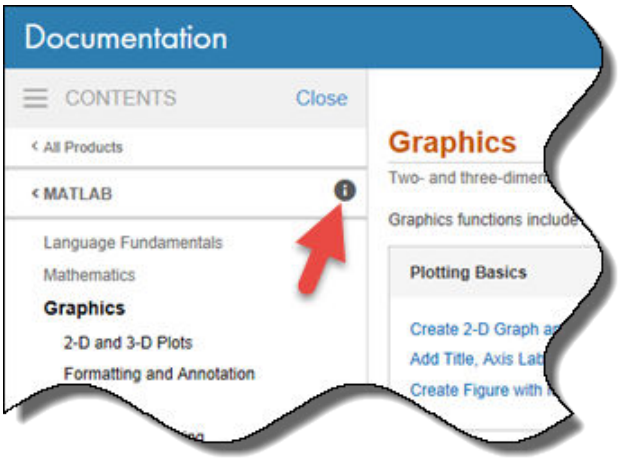
1. On the **Home** tab, select **Add-Ons** > **Manage Add-Ons**.
2. Click the **Import** button.

Documentation: Find information faster with redesigned Help navigation and browser-style keyboard shortcuts

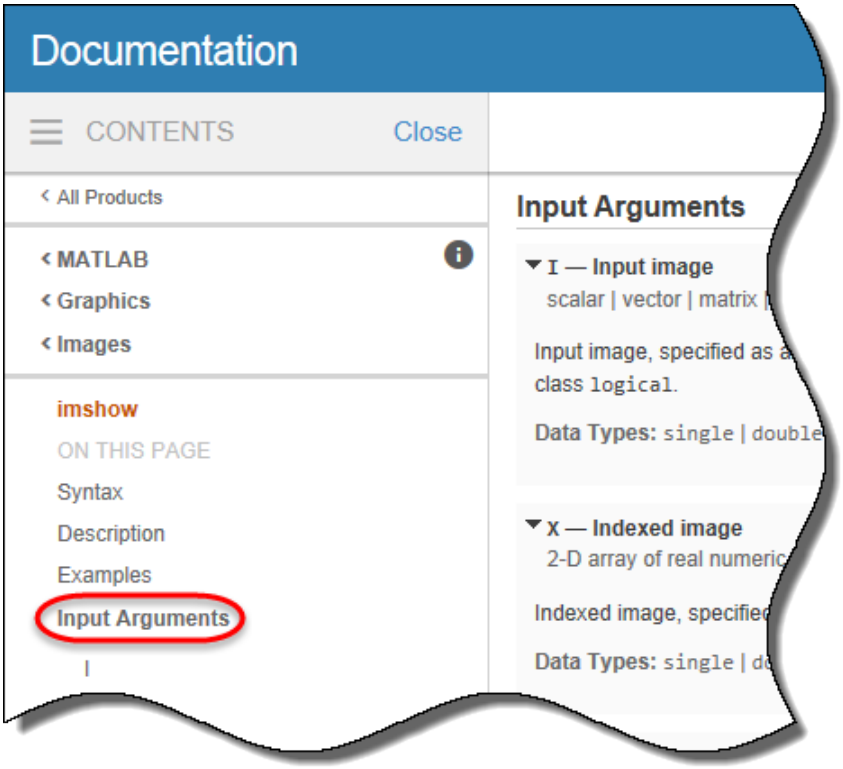
Improved Navigation

The 2015b release includes a redesigned left-side navigation pane to facilitate topic exploration. From this pane you can:

- View the current product, as well as adjacent categories relevant to your current area of focus.
- Access basic product links such as getting started information, examples, functions and other references, release notes, and PDF documentation. Click the  icon to the right of the product name to bring up available options.



- Explore and navigate faster through subsections of a page. Click a subsection title to access its content. For example, suppose you want to view all the input arguments for the `imshow` function. Open the function reference page and in the left-side navigation pane, click **Input Arguments**.

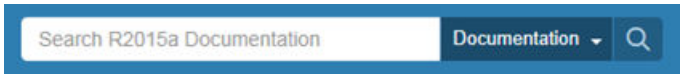


Additional Introductory Material

Product pages and their primary categories now contain enhanced introductory material. New features include product and category overviews, images, quick links, and featured examples.

New Search Bar Location

A redesigned search bar is more visible on every page throughout the documentation. You now can easily find and use the search tool whenever you need it.



New browser-style keyboard shortcuts

New keyboard shortcuts allow better navigation of the MathWorks documentation.

Action	Keyboard Shortcut
Open a new browser tab.	**Ctrl + T**
Open a new browser window.	**Ctrl + N**
Go to the Help browser home.	**Alt + Home**
Reopen the last closed Help browser.	**Ctrl + Shift + T**
Increase or decrease the font size of the Help browser.	**Ctrl + Mouse Scroll**
Open a link in a new tab.	**Ctrl + Mouse Click** or **Mouse Center Click**

Persistent Font Size

Font size now persists over multiple MATLAB Help browser and MATLAB Web browser sessions. To adjust the font size, use the **Ctrl + Mouse Scroll** keyboard shortcut.

Tab Completion: Complete commands with corrected capitalization

MATLAB now ensures command completions have the correct capitalization. For example, if you type `help Contain` and press **Tab**, MATLAB displays `help contains` in the command line.

Command Suggestions: Get suggested corrections for mistyped function names when calling help command

When calling the `help` command, you receive suggested corrections for mistyped function names. Press **Enter** to accept the suggestion or **Esc** to cancel.

Desktop layout: See sharper elements in the MATLAB desktop on high-DPI systems

The appearance of the MATLAB desktop is improved for these system configurations:

- Macintosh systems with Apple Retina displays.
- Windows systems in which the DPI (dots-per-inch) value is higher than 96.

The MATLAB Desktop is now properly scaled and sharper on these systems than in previous releases. Be aware that the icons in the Toolstrip might still look slightly blurry on some systems.

For more information about the DPI-aware changes to MATLAB, see DPI-Aware Behavior in MATLAB.

Compatibility Considerations

- The Toolstrip, Command Window fonts, and Editor fonts might be larger than in previous releases. These differences might occur on Windows systems in which the display DPI value is higher than 96. If you have nondefault font sizes selected in MATLAB preferences, you might need to adjust those font sizes to make them look smaller.
- When publishing a document on a high-DPI system, the images saved to disk are larger than in previous releases or on other systems.

Language and Programming

findgroups and splitapply Functions: Split data into groups and apply functions to each group of data

The `findgroups` function finds groups within grouping variables. Use these groups and `splitapply` to split data out of data variables and apply functions to each group of data.

NaT Function: Create array of Not-a-Time values

The `NaT` function creates an array of Not-a-Time (NaT) values to represent unknown or missing datetime values.

timezones Function: Display list of time zone names

The `timezones` function displays a list of IANA time zone names that you can use as inputs to the `datetime` function.

timeofday Function: Calculate duration on days of Daylight Saving Time (DST) shift

The `timeofday` function accounts for the Daylight Saving Time (DST) shift on days when the shift occurs in time zones that observe DST.

Compatibility Considerations

Before MATLAB release R2015b, the `timeofday` function ignored the Daylight Saving Time (DST) shift on days when the shift occurs.

help Command: Specify a variable as input to the help command instead of specifying the variable class name

You can get help using a variable by specifying it as an input to the `help` command. For example if you have a variable `t = datetime`, calling the command `help t` displays help text for the `datetime` class.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`clear`	Still runs	Depends on usage	The `clear` function no longer clears debugging breakpoints. To clear breakpoints, use `dbclear all`. The `clear` function only clears functions that are not currently running. For example, when you call `clear myFun` while *`myFun`* is running, *`myFun`* is not cleared. Calling `clear all`, `clear classes`, and `clear functions` decreases code performance. For more information and alternatives, see `clear`.

Assign to a nonstructure variable as if it were a structure	Errors	Assign to a field of a new variable, or use the `struct` function.	If a variable exists and is not a structure, and you assign a field as if the variable were a structure, MATLAB throws an error. For example,
			
			`a = 1;`
			`a.b = 2;`
			
			returns an error in MATLAB 8.6 and later:
			
			`Field assignment to a non-structure array ob`
			
			Similarly, MATLAB throws an error for the following code:
			
			`x.y = 1;`
			`x.y.z = 2;`
			
			Instead, you can assign a field to a new variable:
			
			`a = 1;`
			`c.b = 2`
			
			You also can use the `struct` function to replace an existing variable with a structure:
			
			`a = 1;`
			`a = struct('b',2)`

Functionality	Result	Use This Instead	Compatibility Considerations
Invoke anonymous functions with implicit variables in the function body or argument list or implicit calls to nested functions	Errors	Use only explicit variables when constructing anonymous functions. Implicit variables and function calls are often encountered in the functions such as `eval`, `evalin`, `assignin`, and `load`. Avoid using these functions when you construct anonymous functions.	If an anonymous function accesses any variable or nested function that is not explicitly referenced in the argument list or body, MATLAB throws an error when you invoke the function. For example, `f = @() eval('x');` implicitly references x and returns an error in MATLAB 8.6 and later. If an anonymous function references implicit variables or nested functions and is saved in an earlier version of MATLAB, it returns an error in MATLAB 8.6 and later.
Call .p files generated prior to MATLAB 7.5 (R2007b)	Errors	None	Rebuild any P-code files generated with MATLAB 7.4 or earlier using a more recent version of MATLAB.
`depdir`	Errors	`matlab.codetools.requiredFilesAndProducts`	Replace all instances of `depdir` with `matlab.codetools.requiredFilesAndProducts`. However, `matlab.codetools.requiredFilesAndProducts` returns the full path of a required file, including the file name.
`depfun`	Errors	`matlab.codetools.requiredFilesAndProducts`	Replace all instances of `depfun` with `matlab.codetools.requiredFilesAndProducts`. However, `matlab.codetools.requiredFilesAndProducts` does not identify opaque classes, such as Java or COM classes. There is no replacement for this functionality.

Advanced Software Development

MATLAB Interface to Python: Clear Python class definitions with clear classes command, useful when reloading revised Python classes

You can modify a Python module that is already loaded into MATLAB, using the MATLAB `clear` function and the Python `importlib.import_module` and `reload` functions. For an example, see Reload Modified User-Defined Python Module.

MATLAB Interface to Python: Pass a handle to a Python function to other Python functions called from MATLAB

To pass a Python function to a Python function, use the function handle `@` symbol. For an example, see Pass Python Function to Python map Function.

MATLAB Interface to Python: Display error class name in error message

MATLAB displays the class name of a Python error, which provides more information about the error. See Troubleshooting Error Messages.

MATLAB Engine for Python: Call MATLAB functions and objects from Python by connecting to a running session of MATLAB

You can connect the MATLAB Engine for Python to a MATLAB session that is already running. For an example, see Connect Python to Running MATLAB Session.

MEX Compiler Support: Compile MEX files with freely available MinGW-w64 compiler on 64-bit Windows

You can use the MinGW-w64 version 4.9.2 compiler from TDM-GCC to build MEX files and standalone MATLAB engine and MAT-file applications. For more information, see Install MinGW-w64 Compiler.

For an up-to-date list of supported compilers, see the Supported and Compatible Compilers website.

MEX Compiler Support: Compile MEX files with Intel Fortran Composer XE 2013

MATLAB R2015b supports the Intel Fortran Composer XE 2013 compiler with Microsoft Visual Studio 2013 Professional (64-bit) for building MEX files and MATLAB engine applications.

MEX Compiler Support: Discontinued for GNU gfortran

MATLAB no longer supports the GNU gfortran compiler for building MEX files and MATLAB engine applications on Apple Mac platforms.

MEX Compiler Support: To be phased out for Microsoft Visual Studio 2008 SP1 and Apple Xcode 5.1

Support for these compilers will be discontinued in a future release, at which time new versions will be supported.

- Microsoft Visual Studio 2008 SP1 and Microsoft Platform SDK version 6.1
- Apple Xcode 5.1

For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

MEX: Display stack trace in warning message

MATLAB now displays the warning stack when a MEX file displays a warning. You can control the information displayed or suppressed by calling the `warning` function with the `backtrace` mode option before calling the MEX file.

Unit Testing Framework: Improve continuous integration workflows with diagnostics from the TAP plugin and a JUnit-style XML plugin

As of R2015b, the `TAPPlugin` includes diagnostic information in the Test Anything Protocol (TAP) stream. For more information, see `matlab.unittest.plugins.TAPPlugin`.

You can add the `XMLPlugin` to the test runner to produce a JUnit-style XML file. Use the output file to integrate test results with continuous integration systems like Jenkins or TeamCity. For more information, see `matlab.unittest.plugins.XMLPlugin`.

Unit Testing Framework: Customize the test runner with plugins that immediately report finalized results and perform system-wide qualifications

Write plugins that immediately report when the test runner finalizes a test result. A test result is final when no remaining test content can modify the results. Examples of when the test runner might modify previously run test results include when it executes code inside of `TestClassTeardown` methods, or when it tears down shared test fixtures. For more information, see the `reportFinalizedResult` method of `TestRunnerPlugin` and Plugin to Generate Custom Test Output Format.

Write plugins that perform system-wide qualifications of a test suite with the `QualifyingPlugin` interface. For more information, see `matlab.unittest.plugins.QualifyingPlugin`.

Unit Testing Framework: Run tests in parallel when available using the runtests function with the UseParallel option

Run a suite of tests in parallel using the `runtests` function and the Parallel Computing Toolbox. For more information, see the `'UseParallel'` name-value pair argument of `runtests`.

Unit Testing Framework: Detect and prevent warnings issued by tests using the new FailOnWarningsPlugin

Add the `FailOnWarningsPlugin` plugin to the test runner to produce a qualification failure when a test issues a warning. For more information, see `matlab.unittest.plugins.FailOnWarningsPlugin`.

Unit Testing Framework: Create interactive TestCase for specified class

You can create a test case to use and debug your test class interactively using the `TestCase.forInteractiveUse` method. This method accepts a `meta.class` instance that describes a `TestCase` subclass. For more information, see `matlab.unittest.TestCase.forInteractiveUse`.

Compatibility Considerations

You should no longer call the constructor of a `TestCase` subclass to use the instance interactively. Instead, pass the `meta.class` instance into the `forInteractiveUse` method. For example, instead of creating a test case using the constructor of `MyTestCase`:

```
tc = MyTestCase;
```

create a test case using the `forInteractiveUse` method:

```
tc = matlab.unittest.TestCase.forInteractiveUse(?MyTestCase);
```

Unit Testing Framework: Specify subfolders and subpackages for the CodeCoveragePlugin

Create a `CodeCoveragePlugin` that reports on source code contained within subfolders of the specified folders and subpackages of the specified packages. For more information, see `matlab.unittest.plugins.CodeCoveragePlugin`.

Unit Testing Framework: Specify test element name for runtests

You can use the `runtests` function to specify the name of a test suite element. For example, `runtests('ATestFile/aTest')`. The `runtests` function combines test suite element names with test files, test classes, packages, or folders to build and run your test suite. For more information, see `runtests`.

Unit Testing Framework: Validate exact list of causes using Throws constraint

As of R2015b, you can use the `'RespectingSet'` name-value pair argument in the `Throws` constraint to validate that the invocation of a function handle throws an exception with only a specified list of expected causes. The `'RespectingSet'` name-value pair argument has a default value of `false`. To create a `Throws` constraint that is not satisfied if the expected exceptions contain causes that are not specified in the `'CausedBy'` name-value pair, set `'RespectingSet'` to `true`. For more information, see `matlab.unittest.constraints.Throws`.

Calling Methods Behavior Change: Methods must be on the MATLAB path when called

Methods must be on the MATLAB path when called. For example, if you create an object and then change your current folder to a folder from which the method file is not visible, MATLAB returns an error when you call that method.

Compatibility Considerations

Before MATLAB release R2015b, after you create an object, you could call a method of that object even if the method definition was no longer on the MATLAB path. With release R2015b, methods must be on the MATLAB path when called.

numArgumentsFromSubscript Function: Return number of arguments for customized indexing methods

If your class overloads `numArgumentsFromSubscript`, MATLAB calls this method instead of `numel` to determine the number of array elements involved in an indexing operation when the number of elements is greater than one.

If your class overloads `numel`, MATLAB still calls your overloaded `numel` method.

subsref and subsasgn Functions: Compute number of arguments correctly

Before MATLAB release R2015b, MATLAB incorrectly computed the number of arguments expected for outputs from `subsref` and inputs to `subsasgn` for some indexing expressions that return or assign to a comma-separated list.

Compatibility Considerations

With release R2015b, MATLAB correctly computes the values of `nargout` and `nargin` according to the number of arguments required by the indexing expression. For more information, see Modify nargout and nargin for Indexing Methods.

subsasgn Function: Call only with output argument

The `subsasgn` function no longer supports a no output argument syntax that modifies the input argument.

Compatibility Considerations

Before MATLAB release R2015b, `subsasgn` modified the input argument when you did not specify an output argument. With release R2015b, you must assign an output argument or MATLAB returns an error:

```
Error using subsasgn
SUBSASGN must be called with an output.
```

display Function: Use new argument to display name

The `display` function supports a second input argument Use this argument to specify the character array to display instead of the variable name.

Java: Use javaArray function to create zero length arrays

MATLAB supports passing 0-length Java arrays to Java functions. To create such an array, use the `javaArray` function.

Compatibility Considerations

Before MATLAB release R2015b, MATLAB converted 0-length Java arrays to `null`.

Java: Preserve variables and functions in memory when using javaaddpath and javarmpath

The `javaaddpath` and `javarmpath` functions clear the definitions of all Java classes defined by files on the dynamic class path. As of MATLAB release R2015a, the functions do not use the `clear java` command, which also deletes global variables and variables from the MATLAB base workspace, and removes all compiled scripts, functions, and MEX functions from memory.

.NET: Convert some .NET types in System.Object[,] arrays to MATLAB types using the cell function

Use this `cell` function syntax to convert `System.DateTime` and `System.String` data to cell arrays of MATLAB data,

```
A = cell(obj,'ConvertTypes',type)
```

where `obj` is a .NET `System.Object[,]` array, and *`type`* is one of the following strings:

- `{'System.DateTime'}` — Convert `System.DateTime` elements to MATLAB `datetime` elements.
- `{'System.String'}` — Convert `System.String` elements to MATLAB strings.
- `{'all'}` — Convert all supported .NET types to equivalent MATLAB types.

`A` is a cell array, that is the same size as the `obj` array.

For an example, see Read Cell Arrays of Excel Spreadsheet Data.

Source Control Integration: Compare any pair of file revisions

In a file under source control in MATLAB, you can select any pair of file revisions to compare. Also, you can sort revisions by column headers in the Compare to Revision dialog box (for example, by date, revision number, or author). The Compare to Revision dialog box has an improved layout that places revision columns above the details pane, making it much easier to read.

Previously you could compare a revision only with your local file, which made investigation of older changes more difficult. You could not compare between other revisions, and you could not sort revisions.

For details, see Review Changes in Source Control.

Profile: Measure time with different wall-clock types

There are two additional wall-clock options for measuring time while profiling your code. The `'performance'` timer uses the wall-clock time from the clock that the operating system uses to measure performance. As of R2015b, this option is the default timer for profiling. The `'processor'` timer uses the wall-clock time directly from the processor. For more information, type `help profile` at the command prompt.

Compatibility Considerations

As of R2015b, the default timer is `'performance'`. In previous versions of MATLAB, the default profiler timer was `'cpu'`, which measures compute time instead of wall-clock time. You may notice differences in the reported times from previous versions of MATLAB. Also, depending on the code you are profiling, the summary of results can appear different.

system, dos, and ! commands: Run Windows commands without running automatic Microsoft environment commands

On Windows platforms, the MATLAB `system`, `dos`, and `!` commands add the `/D` flag to the startup of the Windows Command Processor (`cmd.exe`) program. The `/D` flag disables the running of these AutoRun strings in Windows registry:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Command Processor`
- `HKEY_CURRENT_USER\Software\Microsoft\Command Processor`

Compatibility Considerations

To preserve the behavior of previous releases, prepend `cmd /C` to your operating system command. For example, in MATLAB, the command,

```
!cmd /C "cd"
```

calls the `cd` command using the AutoRun values.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`matlab.unittest.TestCase` constructor	Errors	`matlab.unittest.TestCase.forInteractiveUse`	Replace all instances of creating a `TestCase` object using the default constructor. Use the `TestCase.forInteractiveUse` static method for interactive, command-line use. When you run tests in the unit testing framework, the test runner creates the instances.
`matlab.lang.ObjectUpdateFailure`	Errors	No replacement required	If changes to a class definition result in an incompatibility that MATLAB cannot resolve with existing objects of that class, MATLAB returns an error. In previous releases, MATLAB converted those objects to object of the `matlab.lang.ObjectUpdateFailure` class. For more information, see Automatic Updates for Modified Classes.
Built-in support for the Microsoft Source Code Control Interface, including `cmopts` `verctrl`	Warns	None	MATLAB no longer includes built-in support for the Microsoft Source Code Control Interface (MSSCCI). Replace this functionality with one of these options: • Download and install the MATLAB integration with MSSCCI add-on. • Use the built-in support for Source Control Integration.
`checkin` `checkout` `customverctrl` `undocheckout`	Warns	None	Replace this functionality with one of these: • Built-in support for Source Control Integration. • Source Control Software Development Kit to create a plug-in for your source control. • MATLAB `system` function and the command-line API for your source control tool to replicate existing functionality.
`actxserver('progid', 'interface', 'IUnknown')`	Still runs	None	R2015b is the last release supporting custom COM interfaces (`IUnknown`).

Mathematics

graph and digraph Classes: Create, analyze, and visualize graphs and networks

The `graph` and `digraph` functions enable creation of undirected and directed graphs.

After creating a graph object, there are numerous functions available for modifying the graph, analyzing its properties, applying search algorithms, and visualizing the structure. For more information, see Graph and Network Algorithms.

histcounts2 Function: Bin bivariate data

The `histcounts2` function sorts bivariate data into 2-D bins with data dependent bin picking and options for bin control and normalization.

deg2rad and rad2deg Functions: Convert between radians and degrees

The `deg2rad` function converts angles from degrees to radians. The `rad2deg` function converts angles from radians to degrees.

datetime Arrays: Interpolate dates and times using interp1 function

The `interp1` function accepts `datetime` arrays as an input argument.

duration Arrays: Perform computations on durations using interp1, eps, cummax, cummin, cumsum, mod, and rem functions

The `interp1`, `eps`, `cummax`, `cummin`, `cumsum`, `mod`, and `rem` functions accept `duration` arrays as an input argument.

expm, logm, and sqrtm Functions: Algorithm upgrades

The algorithms for `expm`, `logm`, and `sqrtm` show increased accuracy, with `logm` and `sqrtm` additionally showing improved performance.

histcounts Function: Use categorical array inputs

The `histcounts` function accepts categorical arrays as an input argument.

Data Import and Export

Excel Spreadsheets: Read and write to the same spreadsheet repeatedly with improved performance of xlsread, xlswrite, readtable, and writetable functions

Repeated calls to the `xlsread`, `xlswrite`, `readtable`, and `writetable` functions show significant performance improvement.

datastore Function: Import data from image collections that are too large to fit in memory as an ImageDatastore

Use the `datastore` function to create an `ImageDatastore`, which can import data from image collections that are too large to fit in memory.

`ImageDatastore` is a compatible datastore for use with `mapreduce`.

datastore Function: Import data from text files with support for additional character encoding schemes with TabularTextDatastore

In addition to UTF-8 and US-ASCII, `TabularTextDatastore` now supports all of the character encoding schemes that the `fopen` function supports, such as, ISO-8859-1, windows-1251, and Shift_JIS. For additional supported encoding schemes, see TabularTextDatastore Properties.

Datastore: Specify and return file paths using Files property

Now when modifying the `Files` property of an existing datastore, each string in the input cell array of strings must contain a full path to the file. Previously, the input cell array could contain folder names or strings with wildcard (*) characters to represent multiple files.

The `Files` property for all file-based datastores is now a column vector. Previously, the `Files` property was a row vector.

Compatibility Considerations

Although modifying the `Files` property of an existing datastore requires full paths to the files, this change does not affect the creation of a datastore using the `datastore` function. When creating a new datastore using the `datastore` function, you can specify the location of the files or folders as a single string or a cell array of strings in the form of a column or row vector. The strings also can contain wildcard characters to represent multiple files.

VideoReader Object: Read streams of data more quickly from video files on Mac

`VideoReader` now reads MPEG-4 and MOV video files more quickly on Mac OS.

readtable and writetable Functions: Specify locale for dates in tables

You can specify the locale that the `readtable` and `writetable` functions use for month and day names. Specify the `DateLocale` name-value pair argument to read month and day names from text files, or to write month and day names to text files or spreadsheets.

Scientific File Format Libraries: Upgrades

Library	Version
NetCDF	4.3.3.1 (upgraded from 4.1.3)

NetCDF Library: Enhancements Due to Upgrade

The upgrade to the NetCDF library version 4.3.3.1 brings these enhancements:

- Many bug fixes in the library
- Fixes for older OPeNDAP servers that do not conform to DAP 2.0 protocol

Low-level reading and writing: Access to additional encoding schemes

The `fopen` function now supports additional character encoding schemes, including EUC-KR, GB2312, GB18030, and KOI8-R. After you call `fopen`, subsequent read and write operations such as `fscanf`, `fprintf`, `fread`, and `fwrite` use the specified character encoding.

webread, websave, and webwrite Functions: Use datetime objects as query parameters

The `webread`, `websave`, and `webwrite` functions can accept input arguments specified as `datetime` objects for requests to web services that accept dates and times as query parameters. To match the date and time format required by the web service you call, specify the `Format` property of a `datetime` object.

webread, websave, and webwrite Functions: Specify array format for query parameters that represent multiple values

To specify the array format for query parameters that represent multiple values, specify the `ArrayFormat` property of a `weboptions` object. Pass the query parameters and the `weboptions` object as input arguments to the `webread`, `websave`, or `webwrite` functions.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`aufinfo`	Errors	`audioinfo`	Replace all instances of `aufinfo` with `audioinfo`.
`auread`	Errors	`audioread`	Replace all instances of `auread` with `audioread`.

Functionality	Result	Use This Instead	Compatibility Considerations
`auwrite`	Errors	`audiowrite`	Write audio files using `audiowrite`.
`wavfinfo`	Errors	`audioinfo`	Replace all instances of `wavfinfo` with `audioinfo`.
`wavread`	Errors	`audioread`	Replace all instances of `wavread` with `audioread`.
`wavwrite`	Errors	`audiowrite`	Replace all instances of `wavwrite` with `audiowrite`.
`movie2avi`	Warns	`VideoWriter`	Remove all instances of `movie2avi`. Write to AVI files with `VideoWriter`.
`datastore(location,'DatastoreType',Value)`	Still runs	`datastore(location,'Type',Value)`	Use name-value name `'Type'` instead of `'DatastoreType'`.
`KeyValueLimit` property of `KeyValueDatastore` class	Warns	`ReadSize` property of `KeyValueDatastore` class	Replace all instances of `KeyValueLimit` with `ReadSize`.
`RowsPerRead` property of `TabularTextDatastore` class	Warns	`ReadSize` property of `TabularTextDatastore` class	Replace all instances of `RowsPerRead` with `ReadSize`.
Folders and wildcard (*) characters in `Files` property of `KeyValueDatastore` and `TabularTextDatastore`	Warns	A string or cell array of strings where each string represents a full path to the file	Replace all folder names or wildcard characters with full path and file names when assigning to the `Files` property.
`hdfgd`	Warns	https://www.mathworks.com/help/releases/R2014b/matlab/ref/matlab.io.hdfeos.gd.html `matlab.io.hdfeos.gd`	Replace all instances of `hdfgd` with the corresponding function in the `matlab.io.hdfeos.gd` package.
`hdfsd`	Warns	https://www.mathworks.com/help/releases/R2014b/matlab/ref/matlab.io.hdf4.sd.html `matlab.io.hdf4.sd`	Replace all instances of `hdfsd` with the corresponding function in the `matlab.io.hdf4.sd` package.
`hdfsw`	Warns	https://www.mathworks.com/help/releases/R2014b/matlab/ref/matlab.io.hdfeos.sw.html `matlab.io.hdfeos.sw`	Replace all instances of `hdfsw` with the corresponding function in the `matlab.io.hdfeos.sw` package.

Graphics

Graphics Display: Leverage DPI-aware MATLAB graphics for Apple Retina displays and high-resolution displays on Windows

The appearance of graphics is improved for these system configurations:

- Macintosh systems with Apple Retina displays
- Windows systems in which the DPI (dots-per-inch) value is higher than 96

Previously, MATLAB allowed the operating system to scale graphical elements on these high-DPI systems. That scaling helped to maintain consistent appearance and functionality, but it also introduced undesirable effects. Graphical elements often looked blurry, and the size of those elements was sometimes inconsistent.

Instead of relying on the operating system, MATLAB is now DPI-aware, which means that it takes advantage of the full system resolution to draw graphical elements (fonts, UIs, and graphics). Thus, graphical elements appear sharp and appropriately scaled regardless of your display DPI. For more information, see DPI-Aware Behavior in MATLAB.

Compatibility Considerations

- In order to display graphics at an appropriate scale, distances in pixels are now device-independent on Windows and Macintosh systems. If you set the `Units` property or `FontUnits` property of an object to `'pixels'`, the size of each pixel is as follows:

  - Windows systems — 1 pixel is 1/96 inch.
  - Macintosh systems — 1 pixel is 1/72 inch.
  - Linux systems — No change. Pixel size determined by display DPI.

  In most cases, you do not need to modify your code. Keep in mind that specifying the size and location of graphics components in pixels might not correspond to the actual pixels on your screen. For example, when the `Units` property of a figure is set to `'pixels'`, MATLAB reports the `Position` and `CurrentPoint` property values based on device-independent pixels, not the size of the actual pixels on the screen.
- MATLAB might report the size and location of objects as fractional values (in pixel units) more frequently than in previous releases. For example, your code might report fractional values in the `Position` property of a figure, whereas previous releases reported whole numbers for that same figure.
- The `ScreenSize` property of the root object might not match the display size reported by high-DPI Windows systems. Specifically, the values do not match when the `Units` property of the root object is set to `'pixels'`. MATLAB reports the value of the `ScreenSize` property based on device-independent pixels, not the size of the actual pixels on the screen.
- When using `getframe` (or `print` with the `-r0` option) on a high-DPI system, the size of image data array that MATLAB returns is larger than in previous releases. Additionally, the number of elements in the array might not match the figure size in pixel units. MATLAB reports the figure size based on device-independent pixels. However, MATLAB bases the size of the image array on the display DPI value.
- The `ScreenPixelsPerInch` property of the root object is now read only, and the value depends on the system:

- Windows systems — The value is 96 DPI.
- Macintosh systems — The value is 72 DPI.
- Linux systems — The value is determined by your display DPI.

Also, you cannot set or query the default value of the `ScreenPixelsPerInch` property. These commands now return an error:

```
get(groot,'DefaultRootScreenPixelsPerInch')
set(groot,'DefaultRootScreenPixelsPerInch')
```

The factory value cannot be queried either. This command returns an error as well:

```
get(groot,'FactoryRootScreenPixelsPerInch')
```

Axes Object: Set the location of each axis in a plot to cross at the origin

Axes objects now have an `'origin'` option for the `XAxisLocation` and `YAxisLocation` properties. Use this option to display the *x*-axis or *y*-axis so that it passes through the origin point `(0,0)`. These properties only affect 2-D views of the axes.

Numeric Ruler Object: Customize tick format, exponent, and text style to control the appearance of an individual axis in a plot

Numeric rulers are new objects that you can use to modify the appearance or behavior of the *x*-axis, *y*-axis, or *z*-axis. When you create an axes object, MATLAB creates a numeric ruler for each individual axis. Modify a particular axis by setting properties of the associated numeric ruler. Access the number ruler through the new `XAxis`, `YAxis`, or `ZAxis` property of the axes object.

Numeric rulers have several properties that offer new types of axis customizations. Control the format of the tick labels using the `TickLabelFormat` property. For example, you can display the tick labels as currency values or control the number of decimal places displayed. Additionally, you can control if the tick labels use exponential notation with the `Exponent` property.

For a list of properties, see Numeric Ruler Properties.

histogram2 Function: Plot bivariate histograms with control over bins, normalization, and display

The `histogram2` function plots bivariate histograms with data-dependent bin picking and options for bin control, normalization, and visualization.

When you specify an output argument with the `histogram2` function, it returns an object that you can use to modify properties of the histogram. For example, you can change the number of bins in each dimension, use a different binning algorithm, or normalize the histogram in several ways. For more information, see Using histogram2 Objects.

histogram Function: Use categorical array inputs

The `histogram` function accepts categorical arrays as an input argument.

`histogram` objects created using categorical data contain some different properties than normal numeric histograms. For more information, see Histogram Properties.

histogram and histogram2 Functions: Interactively manipulate histograms with context menu

Histogram plots created using either `histogram` or `histogram2` have a context menu in plot edit mode that enables interactive manipulations in the figure window.

MATLAB Charts: Use transparency in bar, scatter, and area charts

Area, bar series, and scatter series objects now support transparency. This table lists the new transparency properties for each object. Set these properties to a value between 0 (transparent) and 1 (opaque).

Object	New Properties
Area	`FaceAlpha` `EdgeAlpha`
Bar series	`FaceAlpha` `EdgeAlpha`
Scatter series	`MarkerFaceAlpha` `MarkerEdgeAlpha`

Alternatively, you now can use the `alpha` function to change the transparency of these objects. Specify the input to `alpha` as a value between 0 and 1, `'opaque'`, or `'clear'` to affect these objects.

Contour Plots: Control appearance of contour labels

Control the appearance of contour labels by setting text properties with any `clabel` syntax. In R2014b and R2015a, not all `clabel` syntaxes supported name-value pairs to set text properties.

The changes to `clabel` are as follows:

- All syntaxes now support name-value pairs to set a useful subset of text properties.
- Syntaxes that previously supported name-value pairs to set text properties now support only a subset of text properties.

For the list of supported properties, see `clabel`.

Compatibility Considerations

There are some text properties that you can no longer set using `clabel`. For example, you cannot set the `Parent`, `Children`, or `UserData` properties as name-value pairs. All the syntaxes that are more restrictive in terms of supported text properties continue to return text objects. If necessary, you can use those text objects to set additional properties.

opengl Function: Use a basic version of hardware-accelerated OpenGL

The `opengl` function now supports the `hardwarebasic` option. This option uses your graphics hardware, but disables these graphics features that are unstable with some graphics drivers:

- Align vertex centers feature for sharp vertical and horizontal lines
- Depth peel transparency feature for accurate transparency in overlapping 3-D objects

The disabled features might change in future releases as graphics features change and graphics hardware evolves.

Line Objects: Control style of line corners

Chart lines and primitive lines have a `LineJoin` property for controlling the style of the line corners. Options include round, chamfer, and miter styles.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`plotmatrix` function	Still runs	Not applicable	`plotmatrix` now returns histogram objects instead of patch objects in its fourth output argument. Use histogram properties instead of patch properties to modify the appearance of the histogram plots. For a list, see Histogram Properties.
`ghostscript` function	Errors	None	MATLAB no longer ships with the Ghostscript® library. Remove all instances of code that use `ghostscript.m`.

GUI Building

UI Display: Create DPI-aware UIs for Apple Retina displays and high-resolution displays on Windows

The appearance of UIs have improved for the following high-DPI systems:

- Macintosh systems with Apple Retina displays
- Windows systems in which the DPI (dots-per-inch) value is higher than 96

These are the visual improvements you might see:

- UI components look sharp and render with full graphical detail.
- When you create a UI object, and specify the `Units` as `'pixels'`, the size of that object is now consistent with the size of other objects. For example, the size of a push button (specified in pixels) is now consistent with the size of the text on that push button (specified in points).

For more information about the new DPI-aware behavior, see DPI-Aware Behavior in MATLAB.

Compatibility Considerations

- In order to display UI components at an appropriate scale, distances in pixels are now device-independent on Windows and Macintosh systems. If you set the `Units` property or `FontUnits` property of an object to `'pixels'`, the size of each pixel is as follows:

  - Windows systems — 1 pixel = 1/96 inch
  - Macintosh systems — 1 pixel = 1/72 inch
  - Linux systems — No change. The size of a pixel is determined by the display DPI.

  In most cases, you do not need to modify your code. Keep in mind that specifying the size and location of UI components in pixels might not correspond to the actual pixels on your screen. For example, when the `Units` property of a figure is set to `'pixels'`, MATLAB reports the `Position` and `CurrentPoint` property values based on device-independent pixels, not the size of the actual pixels on the screen.
- MATLAB might report the size and location of objects as fractional values (in pixel units) more frequently than in previous releases. For example, your code might report fractional values in the `Position` property of a figure, whereas previous releases reported whole numbers for that same figure.
- The `ScreenSize` property of the root object might not match the display size reported by high-DPI Windows systems. Specifically, the values do not match when the `Units` property of the root object is set to `'pixels'`. MATLAB reports the value of the `ScreenSize` property based on device-independent pixels, not the size of the actual pixels on the screen.
- UIs might appear larger than in previous releases on Windows systems in which the display scaling value is greater than 100% but less (or equal to) than 125%. Also, icons in those UIs might look slightly blurry because they are enlarged to accommodate the larger UI. To avoid these effects, set the Windows DPI scaling to 100%.

For a list of related changes in Graphics, see “Graphics Display: Leverage DPI-aware MATLAB graphics for Apple Retina displays and high-resolution displays on Windows” on page 11-20.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`menu`	Still runs	`dialog`	The menu function is not recommended. Use the `dialog` function to create multiple choice dialog boxes.

Performance

MATLAB Execution Engine: Run programs faster with redesigned architecture

The new MATLAB execution engine includes performance improvements to function calls, object-oriented operations, and many other MATLAB operations.

These performance improvements result in significantly faster execution of many MATLAB programs with an average speed-up of 40% among 76 performance-sensitive applications from users. Of these tested applications, 13 ran at least twice as fast and only 1 slowed down by more than 10%.

The new execution engine uses just-in-time compilation of all MATLAB code which makes performance more uniform and predictable. The new engine offers improved quality and provides a platform for future performance optimizations and language enhancements.

table Data Type: Access data with improved performance when using dot-parentheses

Using dot-parentheses indexing to access elements of a table variable, for example `x = T.V(I)`, has better performance.

Categorical Arrays: Create a larger number of categories, with improved memory efficiency and performance

- A categorical array can contain a larger number of categories.
- Categorical arrays with small numbers of categories use memory more efficiently.
- Some categorical comparison and assignment operations involving scalars are faster.

Hardware Support

Raspberry Pi 2 Model B: Acquire sensor and image data from Raspberry Pi 2 boards using the MATLAB Support Package for Raspberry Pi Hardware

You can use the MATLAB Support Package for Raspberry Pi Hardware with Raspberry Pi 2 Model B hardware.

BeagleBone Black: Acquire sensor and image data from BeagleBone Black boards using the MATLAB Support Package for BeagleBone Black Hardware

You can use MATLAB commands to connect to a BeagleBone Black board over a network. For more information, see MATLAB Support Package for BeagleBone Black Hardware.

iOS Sensors: Acquire sensor data from Apple iOS mobile devices using the MATLAB Support Package for Apple iOS Sensors

The MATLAB Support Package for Apple iOS Sensors enables you to collect sensor data from your Apple iOS mobile device and log it in MATLAB. You can then process the sensor data in MATLAB in a variety of ways, including creating plots. You can collect data from these sensors:

- Acceleration
- Angular Velocity
- Orientation
- Magnetic Field
- Position

The Apple iOS sensors product requires:

- Installation of MATLAB Mobile™ on your Apple iOS device. You can acquire this application through the App Store.
- Your Apple device must use iOS version 7 or later.
- MATLAB version R2015a or later, for creating the connection to the application and logging sensor data.
- Installation of the MATLAB Support Package for Apple iOS Sensors.

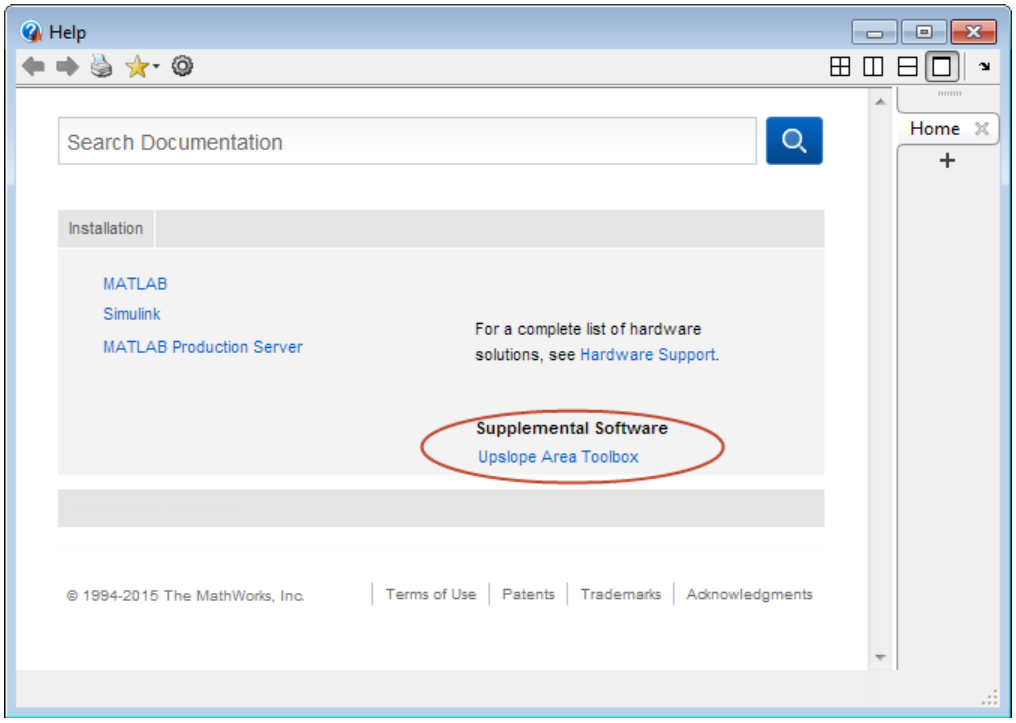
The support package includes the command-line interface in MATLAB and the ability to interact with Apple iOS sensors, which is a separate tab in MATLAB Mobile. To connect to sensors on the phone and collect data, you create a `mobiledev` object in MATLAB. You make the connection between your computer running MATLAB and the phone via Wi-Fi or your cellular network. To get started, see Get Started with Apple iOS Sensors.

R2015a

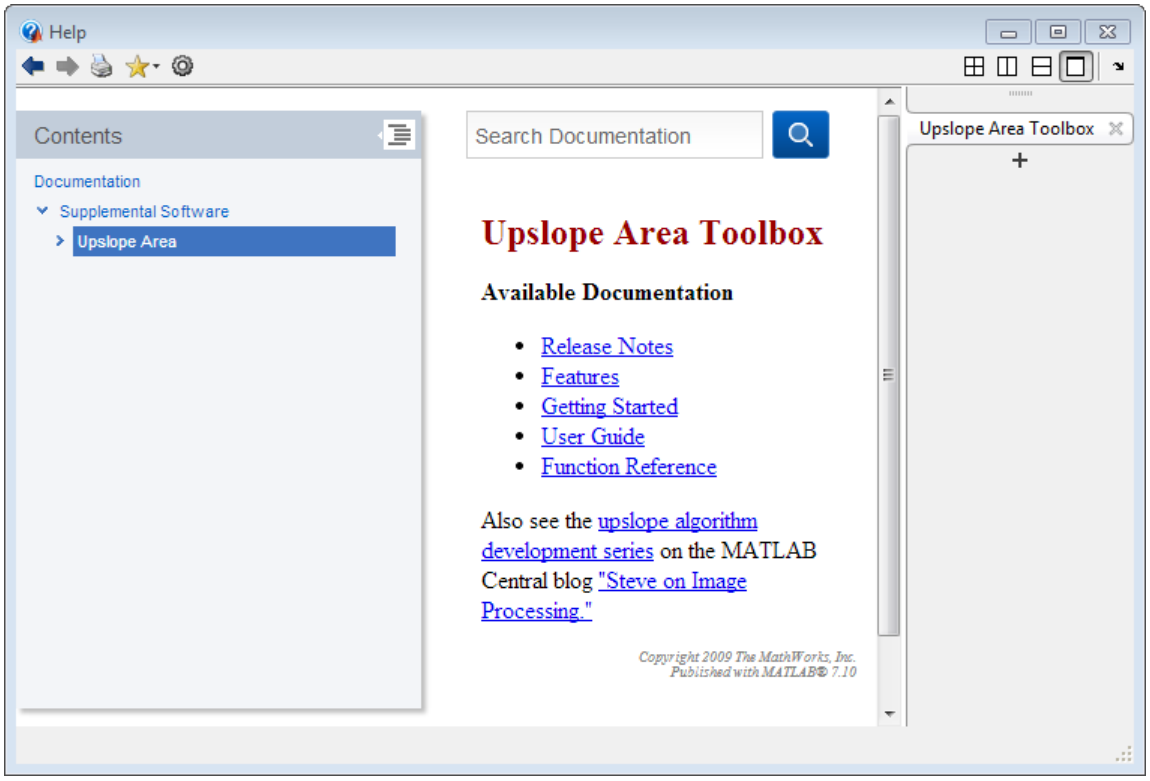
Desktop

Documentation: Integrate documentation for custom toolboxes into the MATLAB Help Browser

When you create custom help and add it to the MathWorks documentation, the help link appears on the Home Help page. For instance, if you add the sample Upslope Area Toolbox documentation, it appears as shown here.



When you click the link for your custom help, that help displays in the current window.



Previously, to access custom help that you added to MathWorks help, you clicked a **Supplemental Software** link at the bottom of the help home page. Then that help opened in a separate window.

For more information on the Upslope Area Toolbox sample and creating custom documentation, see `builddocsearchdb` and Display Custom Documentation.

Documentation: Determine when feature introduced

New documentation reference pages for MathWorks products include an annotation at the bottom of the page. The annotation indicates the release in which the feature was introduced. For example, “Introduced in R2015a.” (Some reference pages for features introduced before R2015a also include the release information.)

Array Size Limit: Limit maximum array size to prevent unintended creation of very large matrices

MATLAB limits the amount of computer memory used to allocate memory for each `mxArray`. To adjust or remove this limit, use the **MATLAB array size limit** preference described in Workspace Browser Preferences.

Tab Completion: Complete class properties and methods while editing class definition files

When editing class definition files in the Editor, you can press the **Tab** key to complete the names of class properties and methods. For more information, see Tab Completion.

User Interface Preferences: Control user interface language

For computers in Japanese, Chinese, or Korean locales, you can choose to run MATLAB with an English user interface rather than in the localized language. Desktop items (such as dialog boxes, button names, and menu items) and error and warning messages will appear in English. For information about setting this option, see General Preferences.

Language and Programming

repelem Function: Repeat copies of array elements to create a larger array

The `repelem` function copies elements of an array into a new array according to a specified repetition scheme.

sort Function: Now preserves shape of cell array of string inputs

The `sort` function now preserves the shape of input cell arrays of strings and sorts each column independently. This makes the behavior for `sort` with a cell array of strings consistent with the behavior of `sort` with a numeric array.

Compatibility Considerations

Previously, `sort` flattened all cell array of string inputs into a vector output. This behavior treated the input cell array as a tall vector of strings, and did not preserve the size of the input. Any code that relies on the output size of `sort` on a cell array of strings might need to be updated.

To continue using the previous behavior, use the colon operator, `:`, when calling `sort` on a cell array, for example, `sort(X(:))`.

isenum Function: Determine if variable is enumeration

The `isenum` function tests the input argument to determine if it is an enumeration.

milliseconds Function: Convert duration to number of milliseconds

The `milliseconds` function converts `duration` values to `double` values in units of milliseconds.

Publishing Markup: Include external file content

External file content can be added into MATLAB published code using the new `<include>` markup. For information about this new markup, see External File Content.

fullfile Function: Maintain all double-dot symbols

The `fullfile` function now maintains all double-dot symbols in the input file parts. This behavior is consistent with the treatment of double-dot symbols in R2012b and earlier.

In R2013a through R2014b, `fullfile` collapsed inner relative directories indicated by double-dot (..) symbols and returned an equivalent full file path specification that did not display the symbols. The double-dot symbols were maintained only if they appeared at either end of the full file path specification.

For example, the commands,

```
f = fullfile('C:\foo\folder1\..\folder2')
h = fullfile('\\folder4\.')
```

now return:

```
f =
C:\foo\folder1\..\folder2
h =
\\folder4\.
```

In R2013a through R2014b, the same commands returned:

```
f =
C:\foo\folder2
h =
\\folder4\.
```

Compatibility Considerations

Change code that relies on resolving double-dot symbols in a full file path.

Python Objects: Indexing Support

For more information, see:

- Accessing Elements in Python Container Types
- Limitations to Indexing into Python Objects

For examples, see:

- Index into Python String
- Index into Python list
- Index into Python tuple
- Index into Python dict

Python Version 3.4: MATLAB Support

MATLAB supports the following versions of CPython:

- Version 2.7
- Version 3.3
- Version 3.4

For more information, see Install Supported Python Implementation.

MATLAB Engine for Python: Support for startup options

The engine function `matlab.engine.start_matlab` supports MATLAB startup options. In addition, the engine supports `'-desktop'` to launch MATLAB with the desktop.

MATLAB Engine for Python: Support for Unicode in Python 2.7

From Python 2.7 you can pass variables of data type `unicode` as input arguments to MATLAB functions. The engine can return `unicode` output arguments to Python 2.7.

Conversion of Character Arrays to Java Strings: Preserve null characters

In previous versions of MATLAB, the Java string was terminated at the first null.

WSDL Web Services Documents: Limitations

When you communicate with web services using web services description language (WSDL) documents, errors can occur if a WSDL file imports another WSDL file that uses the WSDL import element. For more information, see Limitations to WSDL Document Support.

Unit Testing Framework: Tag tests for categorization and selection

You can use tags to group tests into categories, and then run tests with specified tags. Tag your tests using the class-level or method-level `TestTags` attribute on `TestCase`. Select tagged test elements using the `HasTag` selector or the `'Tag'` name-value pair on one of the following `TestSuite` methods: `fromClass`, `fromFile`, `fromFolder`, `fromMethod`, `fromPackage`, or `selectIf`. You also can select and run tagged tests using the `'Tag'` name-value pair with the `runtests` function. For more information, see Tag Unit Tests.

Unit Testing Framework: Run tests in parallel

You can run a suite of tests in parallel using the Parallel Computing Toolbox. For more information, see the `runInParallel` method of the `TestRunner` class.

Unit Testing Framework: Share variables between tests in scripts

You can include a shared variables section in your script-based test file. In this section you can define variables to share between tests in the script and define any critical preconditions necessary for your tests. For more information, see https://www.mathworks.com/help/releases/R2015a/matlab/matlab_prog/write-script-based-unit-tests.html.

Unit Testing Framework: Use prebuilt test fixtures

You can manually configure your test environment, and the test runner does not need to perform shared fixture setup and teardown activities. To define prebuilt fixtures, use the `PrebuiltFixtures` property on the `TestRunner` instance in combination with the `SharedTestFixtures` attribute of the `TestCase` class. In certain testing scenarios, prebuilding fixtures can result in significant time or resource savings. An example where prebuilt test fixtures might be a benefit is a test that meets these three conditions:

- Has shared fixtures that are expensive to set up
- Has a required fixture state that can be set up manually in the MATLAB session
- Does not require a fresh test fixture for each test run

For more information, see `matlab.unittest.TestRunner`.

Unit Testing Framework: Compare objects using isequaln

When the class of the expected value defines an `isequaln` method, the `ObjectComparator` uses that method for object comparison.

In previous releases, `ObjectComparator` used `isequal` to compare all objects. The `IsEqualTo` constraint and the `assertEqual`, `assumeEqual`, `fatalAssertEqual`, and `verifyEqual` qualification methods leverage `ObjectComparator` and, therefore, inherit the same change in behavior. For more information, see `ObjectComparator`.

Compatibility Considerations

Rewrite your tests to force a comparison with `isequal` when you use `ObjectComparator` to compare objects of a class that defines `isequaln` and you rely on the results of `isequal` instead of `isequaln`.

For example, if the class of `expObj` defines `isequaln`, then `testCase.verifyEqual(actObj,expObj)` uses `isequaln` for comparison, but `testCase.verifyReturnsTrue(@()isequal(actObj,expObj))` uses `isequal`.

Similarly, `testCase.verifyThat(actObj,IsEqualTo(expObj))` uses `isequaln` for comparison, but `testCase.verifyThat(@()isequal(actObj,expObj),ReturnsTrue)` uses `isequal`.

Unit Testing Framework: Use homogeneous expected causes with Throws constraint

When you specify expected causes using the `'CausedBy'` name-value pair argument in the `Throws` constraint, the causes must be a cell array of strings or an array of `meta.class` instances.

In previous releases, you specified the value of `'CausedBy'` as a cell array that could contain both strings and `meta.class` instances. For more information, see `matlab.unittest.constraints.Throws`.

Compatibility Considerations

Rewrite tests that use the `Throws` constraint and specify heterogeneous data types for `'CausedBy'`. If you cannot define expected causes using either a cell array of strings or an array of `meta.class` instances, consider performing the qualification using two separate `Throws` constraints.

Git Source Control Integration: View branch details and delete branches

When using the Git source control integration, you can view branch details and delete branches in the Manage Branches dialog box. Use the **Branch Browser** to see which files changed for a particular commit in the branch or to view the author, date, and commit message.

For more information, see Branch and Merge with Git.

C Matrix Library: New functions

The new functions `mxCreateUninitNumericMatrix` and `mxCreateUninitNumericArray` create numeric arrays without initializing the data elements. If you want to initialize the data elements, use the functions `mxCreateNumericMatrix` and `mxCreateNumericArray`.

To convert an `mxArray` to a C-style string in UTF-8 encoding, call the new `mxArrayToUTF8String` function.

To determine if an `mxArray` contains scalar data, call the new `mxIsScalar` function.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`isglobal`	Errors	none	Refactor any code that uses `isglobal`. Avoid conditions in your code that check variable scope. If you need to convert your code quickly in the short term, replace `isglobal` with `~isempty(whos('global','variable'))`. However, you should refactor your code to avoid conditional checks on variable scope.
`matlab.unittest.TestCase` constructor	Warns	`matlab.unittest.TestCase.forInteractiveUse`	Replace all instances of creating a `TestCase` object using the default constructor. Use the `TestCase.forInteractiveUse` static method for interactive, command line use. When you run tests in the unit testing framework, the test runner creates the instances.
`verctrl`	Still runs	Source control action through context menu	Access source control actions through the context menu instead of using `verctrl`.
`cmopts`	Still runs	Click the **Preferences** button on the **Home** tab, and select **General** > **Source Control**.	View the currently selected source control system through **Preferences** instead of using `cmopts`.

Functionality	Result	Use This Instead	Compatibility Considerations

`checkin`
`checkout`
`customverctrl`
`undocheckout`
	Still runs	none	Replace this functionality with one of the following options.
			• Use the built-in support for Source Control Integration.
			• Use the Source Control Software Development Kit to create a plug-in for your source control.
			• Use the MATLAB `system` function and the command-line API for your source control tool to replicate existing functionality.

Mathematics

discretize Function: Group numeric data into bins or categories

The `discretize` function groups numeric data into bins. By default the bins are numeric, and `discretize` returns indices describing the bin placement of each element. However, the bins also can be categories, in which case `discretize` returns a categorical array.

Descriptive Statistics: Omit NaN values in basic statistical calculations, including max, min, mean, median, sum, var, std, and cov

The following functions now have flags to include or omit `NaN` values in numeric computations: `max`, `min`, `mean`, `median`, `sum`, `var`, `std`, and `cov`.

ismembertol and uniquetol Functions: Perform set comparisons using a tolerance

The `ismembertol` and `uniquetol` functions perform set operations using a tolerance. These functions are useful in situations where floating-point round-off errors occur in calculations.

Random Numbers: Generate random numbers using the double-precision, SIMD-oriented Fast Mersenne Twister (dSFMT) algorithm

The dSFMT random number generator algorithm provides a faster implementation of the Mersenne Twister algorithm. For more information about available random number generator algorithms, see Creating and Controlling a Random Number Stream.

nearestNeighbor Function: Determine nearest alphaShape boundary point

The `nearestNeighbor` function for `alphaShape` determines the nearest boundary point on the alpha shape for a specified query point.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`bitmax`	Errors	`flintmax`	Replace all instances of `bitmax` with `flintmax`.

Data Import and Export

Datastore: Read one complete file with 'file' option for ReadSize property

The `ReadSize` property for `TabularTextDatastore` objects now accepts the string value, `'file'`. This option reads one complete text file at a time. For more information, see TabularTextDatastore Properties.

Datastore: Read data in parallel from a datastore with partition function using Parallel Computing Toolbox

The `partition` function partitions a datastore into a number of parts. You then can specify a subset of the datastore to read. If you have Parallel Computing Toolbox installed, you can read datastore parts in parallel without using MapReduce.

webwrite Function: Send data to RESTful Web services using HTTP POST method

The `webwrite` function sends data to RESTful Web services using an HTTP POST request.

webread and websave Functions: Request data from RESTful Web services using HTTP POST method

The `webread` and `websave` functions can request data using both HTTP GET and POST methods.

xlsread and readtable Functions: Read larger spreadsheet files from Excel software

The `xlsread` and `readtable` functions can read larger spreadsheet files on Windows computers with Excel software installed.

textscan and readtable Functions: Return consistent results when reading quoted strings

As of R2014b, using the `%q` format specifier with the `textscan` and `readtable` functions returns consistent results. When using `%q` to read strings that begin with a double quotation mark (`"`), `textscan` and `readtable` remove the leading double quotation mark and its accompanying closing mark. The closing mark is the second instance of a lone, unescaped double quotation mark. The functions replace escaped double quotation marks (for example, `""abc""`) with lone double quotation marks (`"abc"`). The `textscan` and `readtable` functions ignore any double quotation marks that follow the closing mark.

In R2014a and earlier, the `textscan` and `readtable` functions return unpredictable results when reading strings with extraneous double quotation marks. This behavior can occur when you use the `%q` format specifier or if you do not specify a value for the `Format` name-value pair argument to `readtable`.

For example, these commands:

```
C = textscan('"Alvin ""Al"" Smith"','%q','Delimiter',',');
C1 = C{:}
D = textscan('"Alvin" "Al" Smith"','%q','Delimiter',',');
D1 = D{:}
E = textscan('"Alvin ""Al"" Smith" and Brian"""Jones','%q','Delimiter',',');
E1 = E{:}
```

now return:

```
C1 =
    'Alvin "Al" Smith'
D1 =
    'Alvin "Al" Smith"'
E1 =
    'Alvin "Al" Smith and Brian"""Jones'
```

In R2014a and earlier, the same commands return:

```
C1 =
    'Alvin "Al" Smith'
D1 =
    'Alvin "Al Smith"'
E1 =
    'Alvin "Al" Smith and Brian""Jones'
```

Compatibility Considerations

Code that relies on using the `%q` format specifier to read text enclosed in double quotation marks might return different results. Such code includes calls to `readtable` that do not specify a value for the `Format` name-value pair argument. However, there is no behavior change when reading strings with proper double quotation marks.

Scientific File Format Libraries: Upgrades

This table lists upgrades to scientific file format libraries used by MATLAB.

Library	Version
HDF5	1.8.12 (upgraded from 1.8.6)

Compatibility Considerations

The `H5Pset_dxpl_multi` and `H5Pget_dxpl_multi` routines were removed from the HDF5 Library as of version 1.8.11. As a result, the `H5P.set_dxpl_multi` and `H5P.get_dxpl_multi` functions in MATLAB now return an error. Remove instances of `H5P.set_dxpl_multi` and `H5P.get_dxpl_multi` from your code.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`KeyValueLimit` property of `KeyValueDatastore` class	Still Runs	`ReadSize` property of `KeyValueDatastore` class	Replace all instances of `KeyValueLimit` with `ReadSize`.
`RowsPerRead` property of `TabularTextDatastore` class	Still Runs	`ReadSize` property of `TabularTextDatastore` class	Replace all instances of `RowsPerRead` with `ReadSize`.
`urlread` for HTTP POST requests	Still runs	`webread` `webwrite`	Use `webread` for reading content with an HTTP POST request. Use `webwrite` for sending data with an HTTP POST request.
`H5P.get_dxpl_multi`	Errors	none	Remove all instances of `H5P.get_dxpl_multi`.
`H5P.set_dxpl_multi`	Errors	none	Remove all instances of `H5P.set_dxpl_multi`.

Graphics

drawnow Function: Improve performance in animation loops with new option

`drawnow` now supports the `limitrate` option for limiting the number of figure updates to 20 frames per second. If you are updating graphics objects in a loop and do not need to see every update on the screen, you can use this option to skip some updates for a faster animation.

Functionality being removed or changed

Functionality	Result	Use Instead	Compatibility Considerations
`drawnow update` syntax	Still runs	`drawnow limitrate`	Replace all instances of `drawnow update` with `drawnow limitrate`.
`drawnow expose` syntax	Still runs	`drawnow nocallbacks`	Replace all instances of `drawnow expose` with `drawnow nocallbacks`.

Performance

MapReduce: Run mapreduce algorithms on any computer cluster that supports parallel pools using MATLAB Distributed Computing Server

The `mapreduce` function now runs on any cluster that supports a parallel pool.

For more information about `mapreduce` execution environments, see Speed Up and Deploy MapReduce Using Other Products.

Interpolation Functions: Execute faster with multithreaded calculations

The `griddedInterpolant`, `interp1`, `interp2`, `interp3`, and `interpn` functions show significant performance improvement by using multithreading in large calculations.

Hardware Support

IP camera: Acquire video directly from Internet Protocol cameras

You can use the MATLAB IP Camera support to bring live images into MATLAB Images can be from any IP Camera that supports Motion JPEG stream over HTTP or RTSP with basic authentication support. IP cameras, also referred to as netcams or network cameras, are increasingly popular in live image processing applications.

With simple MATLAB functions you can preview your image, acquire single snapshots from the camera, and optionally set up a loop of acquired images. The `ipcam` function creates the IP Camera object that is used to acquire images, and the `snapshot` function returns a single image from the camera.

IP Camera support is available only through a hardware support package. Download and install the necessary files using the Support Package Installer, which you open by typing `supportPackageInstaller` in MATLAB. On the **Select support package to install** screen, select `IP Cameras` from the list.

You can use MATLAB IP Camera support on the following platforms:

- Microsoft Windows 32-bit and 64-bit
- Mac OS X 64-bit
- Linux

For more information about the IP camera support, see the MATLAB Support Package for IP Cameras documentation, which is available when you download the support package.

BeagleBone Black Hardware: Access BeagleBone Black hardware with the MATLAB Support Package for BeagleBone Black Hardware

You can use MATLAB commands to connect to a BeagleBone Black board over a network and perform the following operations:

- Exchange data with sensors and actuators that are connected to the ADC, GPIO, I2C, PWM, serial port, and SPI interfaces
- Take photographs using a web camera
- Issue Linux shell commands
- Transfer files to or from your host computer
- Control the onboard LED

Using the command-line interface in MATLAB, you can:

- Start programing without additional toolboxes.
- Interactively develop and debug programs in MATLAB.
- Acquire and process sensor data in MATLAB in a variety of ways, including creating plots.

For more information, see MATLAB Support Package for BeagleBone Black Hardware.

Arduino Hardware: Access to Arduino Leonardo and other boards with the MATLAB Support Package for Arduino Hardware

You can now use these boards with MATLAB Support Package for Arduino Hardware:

- Arduino Leonardo
- Arduino Micro
- Arduino Mega ADK
- Arduino Nano 3.1
- Sparkfun Redboard
- Sparkfun Digital Sandbox
- Sainsmart Uno
- Sainsmart Mega 2560

Arduino Hardware: New configurePin function

In release R2015a, use `configurePin` to set your pin mode. You can use string values, such as `'D2'` to specify digital pins.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`targetinstaller`	Warns	`supportPackageInstaller`	Replace all instances of `targetinstaller` with `supportPackageInstaller`.
`configureAnalogPin`	Warns	`configurePin`	Replace all instances of `configureAnalogPin` with `configurePin`.
`configureDigitalPin`	Warns	`configurePin`	Replace all instances of `configureDigitalPin` with `configurePin`.

R2014b

Desktop

Git and Subversion source control system integration through Current Folder browser, including syncing from Web-hosted repositories such as those on GitHub

The MATLAB Current Folder browser includes a column with source code status for files that are contained in Git and Subversion® repositories. You can retrieve repositories and access source control functionality through the context menus. For more information, see Source Control Integration.

Packaging of custom MATLAB toolboxes into a single, installable file

You can package your toolbox as a single installer file (`.mltbx`) that contains all of the code, data, apps, documentation, and examples necessary to use your toolbox. Share your toolbox with others by uploading the installer file to the File Exchange or by sending it as an email attachment. For more information, see Create and Share Toolboxes.

Dialog box for managing custom MATLAB toolboxes

You can use the Manage Custom Toolboxes dialog box to view details about the installed custom toolboxes or to uninstall the toolboxes. To access the Manage Custom Toolboxes dialog box, go to the **Resources** section of the **Home** tab and select  **Manage Custom Toolboxes** from the **Add-Ons** menu.

Preference for controlling the initial working folder, with the option to start in the folder from your previous MATLAB session

For information about setting this option, see General Preferences.

Compatibility Considerations

On Windows platforms, do not use the **Start in** field in the MATLAB Properties window. Instead, use the **Specify the full path to a folder** option in the Preferences panel.

Copying and pasting variables in the Workspace browser

You now can copy and paste the contents of workspace variables in the Workspace browser. Just right-click, select **Copy**, and then **Paste**. Alternatively, you can use the **Ctrl+C** and **Ctrl+V** keyboard shortcuts. Previously, this action opened the Import Tool for importing the variable name as a string.

Self-paced eLearning available from within MATLAB

MATLAB Academy is the entry point to self-paced eLearning from within MATLAB. Through MATLAB Academy, you can participate in interactive training courses to get you started with MATLAB.

To access MATLAB Academy from within MATLAB, go to the **Resources** section of the **Home** tab and select  **MATLAB Academy** from the **Help** menu. Alternatively, you can access MATLAB Academy

from the Getting Started with MATLAB documentation or online at https://matlabacademy.mathworks.com. You must have a MathWorks account associated with an active license.

New startup switch to opt out of automatically switching to software OpenGL

If MATLAB detects it is running with a graphics driver that has known issues, it automatically switches to software OpenGL. You can opt out of this behavior by calling MATLAB with the `-nosoftwareopengl` startup option. See `matlab (Windows)` or `matlab (UNIX)`.

To re-enable automatic switching to software OpenGL, call MATLAB with the `-softwareopengl` startup option.

These startup options are not available on Mac platforms.

Color settings preferences in Comparison Tool

You can change and save your color preferences for the MATLAB Comparison Tool. Apply your color preferences when comparing text files, MAT-files, variables, model files, zip files, or folders. For details, see Change Color Preferences.

Automatic file saving when you click away from the Editor

If you are editing a file in the MATLAB Editor that you have saved at least once, new changes are automatically saved if you click away from the Editor.

This setting is on by default, and is accessible through MATLAB Editor/Debugger Preferences. On the **Home** tab, in the **Environment** section, click **Preferences**, and select **Editor/Debugger**. Disable the setting under **Automatic file changes** by clearing the **Save changes upon clicking away from a file** check box.

Language and Programming

datetime, duration, and calendarDuration arrays for efficient computation, comparison, and formatted display of dates and times

`datetime`, `duration` and `calendarDuration` are new data types for representing dates and times. You can manipulate arrays of these types in the same way that you work with numeric arrays. For example, you can add, subtract, sort, compare, concatenate, and plot date and time values.

The new data types support the following features:

- Functions for both calendar computations and fixed-length time computations on absolute dates and times, and elapsed times
- `TimeZone` property for setting and converting `datetime` time zone, accounting for daylight saving time
- Customizable default display format for dates
- Nanosecond precision for absolute times
- Import of data as `datetime` arrays using the Import Tool and the `readtable` and `textscan` functions

For more information, see Dates and Time.

Compatibility Considerations

When importing data from spreadsheets or text files using the Import Tool, there is no longer an option to convert dates to numeric serial date numbers. Instead, import the data into a table or as column vectors, and specify the `datetime` data type for each column of dates.

Suggested corrections for syntax errors in the Command Window

MATLAB suggests corrections for function names mistyped in the Command Window. This functionality now includes suggestions for:

- Missing closing parentheses or brackets such as `)`, `]`, and `}` in function calls and arrays
- Common assignment operators such as `++`, `+=`, and `-=`, and mathematical operator idioms such as `2(x+1)`
- Python argument syntax and dictionary syntax

Suggested MathWorks products for undefined functions

MATLAB suggests which MathWorks product is required for an undefined function. This information includes links to documentation and product information. For example, if you do not have the Aerospace Toolbox,

```
angle2quat
```

```
To use 'angle2quat', you might need:
  angle2quat - Aerospace Toolbox
```

Prior to R2014b, MATLAB displayed an ‘Undefined function’ error.

Create multiple search indexes for help files you create

You can create multiple search indexes for help files you create: one for when you use MATLAB R2014a or earlier, and another for when you use MATLAB 2014b. MATLAB automatically uses the appropriate index for searching your documentation database. For details, see `builddocsearchdb`

py package for using Python functions and objects in MATLAB, and an engine interface for calling MATLAB from Python

For information about calling Python functions in MATLAB, see Call Python Libraries.

For information about calling MATLAB from Python, see MATLAB Engine for Python.

matlab.wsdl.createWSDLClient function for accessing SOAP-based Web services

For information about using `matlab.wsdl.createWSDLClient`, see Access Services That Use WSDL Documents.

You must download supported versions of the Oracle Java JDK and the Apache CXF programs. For information, see Set Up WSDL Tools.

RPC-encoded WSDL documents are not supported. For these documents, use `createClassFromWsdl`.

The following WSDL documents are not supported:

- Some documents with messages containing multiple parts.
- Some documents with schemas containing anonymous complex types.
- Documents that the Apache CXF program cannot compile into complete code.

Compatibility Considerations

`createClassFromWsdl`, `createSoapMessage`, `callSoapService`, and `parseSoapResponse` will be removed in a future release. Use `matlab.wsdl.createWSDLClient` instead.

Graphics objects in MEX-files use object handles instead of numeric handles

Graphics objects use object handles of various types instead of the numeric handles used in previous releases.

Compatibility Considerations

Do not write MEX-file code using the `mexGet` or `mexSet` functions, which assume a handle to a graphics object is a numeric value. Use the `mxGetProperty` or `mxSetProperty` functions instead. For more information, see Upgrade MEX-Files to Use Graphics Objects.

Workflow improvements when editing classdef files, including immediate impact on existing and new workspace variables

This feature eliminates the need to clear existing objects or call `clear classes` when changing the definition of the object's class. This feature also provides the ability to create objects based on the new definition without generating a warning that the class has changed. MATLAB updates existing objects according to the new class definition. For more information, see Automatic Updates for Modified Classes.

MATLAB errors attempting to define listener for nonobservable property

In previous releases, you could define a listener for a nonobservable property. MATLAB silently ignored the listener and did not call the listener callback. With release MATLAB 2014b, attempting to create a listener using `event.proplistener` or `addlistener` for a nonobservable property causes an error.

Compatibility Considerations

If your code used meta-data or the `properties` function to get a list of all properties defined by a class, and then assigns listeners to all properties, you should change your code. Rewrite the code to determine if a given property is observable before assigning the listener. For example, this code determines which properties of `MyClass` are `SetObservable` or `GetObservable`.

```
mc = ?MyClass;
p = findobj(mc.PropertyList,'SetObservable',1,'-or','GetObservable',1)
```

`p` contains the `meta.property` objects for the properties of `MyClass` that are `SetObservable` or `GetObservable`.

Script-based tests in unit testing framework

The MATLAB unit testing framework now provides script-based writing, execution, and verification of tests as an alternative to writing function-based or class-based tests. For more information, see Write Script-Based Unit Tests.

Plugin to report code coverage in unit testing framework

The `matlab.unittest` testing framework provides a plugin that produces a code coverage report for MATLAB source code. For more information, see the `matlab.unittest.plugins.CodeCoveragePlugin` documentation.

Control logging and verbosity in unit testing framework

Test authors can control the verbosity of logged messages in their tests. To register messages at different verbosity levels, call the `log` method with a `TestCase` or `Fixture`. To control the verbosity of the output, use a plugin such as `LoggingPlugin` or `TestRunProgressPlugin`, or a test runner constructed with the `TestRunner.withTextOutput` method.

Constraint for scalar values in unit testing framework

You can use the `matlab.unittest.constraints.IsScalar` constraint class to test that an actual value is a scalar.

Test suites from packaged functions and scripts

The unit testing framework recognizes function-based and script-based tests that are contained within a user-defined package. For more information, see `TestSuite.fromPackage` and `runtests`.

Failure of unit tests using a relative tolerance when the expected value is infinite and the actual value is finite

If a test qualification applies a relative tolerance and the expected value is infinite, the test fails if the actual value is finite. For example,

```
tc = matlab.unittest.TestCase.forInteractiveUse;
tc.verifyEqual(1,Inf,'RelTol',1e-12);
```

```
Interactive verification failed.

---------------------
Framework Diagnostic:
---------------------
verifyEqual failed.
--> The values are not equal using "isequaln".
--> The error was not within relative tolerance.
--> Failure table:
            Index    Actual    Expected    Error    RelativeError    RelativeTolerance
            _____    ______    ________    _____    _____________    _________________

            1        1         Inf         -Inf     NaN              1e-12

Actual double:
         1
Expected double:
       Inf
```

Compatibility Considerations

Change any instances of test qualifications in which you compare an infinite expected value to a finite actual value using a relative tolerance, and expect the test to pass.

rmdir treatment of asterisk as literal character on Linux and Mac

On Linux and Mac platforms, the `rmdir` function treats the asterisk character (*) as a literal character if a file or folder named * exists in the current folder. If such a file or folder does not exist, then `rmdir` treats the asterisk as a wildcard character. On Windows, `rmdir` always treats * as a wildcard character.

Previously, `rmdir` treated * as a wildcard character on all platforms.

Compatibility Considerations

Change code that expects to remove all files and folders using a wildcard character, on Linux and Mac.

Functionality being removed or changed

Functionality	Result	Use This Instead	Compatibility Considerations
`createClassFromWsdl`	Warns	`matlab.wsdl.createWSDLClient`	Replace all instances of `createClassFromWsdl` with `matlab.wsdl.createWSDLClient`. RPC-encoded WSDL documents are not supported.

`createSoapMessage`
`callSoapService`
`parseSoapResponse`
	Still runs	`matlab.wsdl.createWSDLClient`	Replace all instances of `createClassFromWsdl` with `matlab.wsdl.createWSDLClient`.

`matlab.unittest.TestCase` constructor	Still runs	`matlab.unittest.TestCase.forInteractiveUse`	Replace all instances of creating a `TestCase` object using the default constructor. Use the `TestCase.forInteractiveUse` static method for interactive, command line use. When tests are run in the unit testing framework, instances are created by the test runner.
`matlab.unittest.plugins.TestSuiteProgress-Plugin`	Still runs	`matlab.unittest.plugins.TestRunProgressPlugin`	Replace all instances of `TestSuiteProgressPlugin`. To construct a plugin with the same level of detail, use `TestRunProgressPlugin.withVerbosity(2)`.

Functionality	Result	Use This Instead	Compatibility Considerations
`mexGet` `mexSet`	Errors	`mxGetProperty` `mxSetProperty`	For MEX-files, replace all instances of `mexGet` and `mexSet` with `mxGetProperty` or `mxSetProperty` in the C/C++ and Fortran Matrix Library.

Mathematics

histcounts function for binning numeric data

The `histcounts` function sorts data into bins with data-dependent bin picking and options for bin control and normalization.

triangulation functions nearestNeighbor and pointLocation for identifying the closest vertex and enclosing triangle or tetrahedron for specified point

The `nearestNeighbor` function identifies the closest vertex to a query point, and the `pointLocation` function identifies the enclosing triangle or tetrahedron for a query point.

Option for interpolating to 'next' and 'previous' neighbors with the interp1 function and griddedInterpolant class

The `interp1` function and `griddedInterpolant` class now have 1-D support for interpolating to `'next'` and `'previous'` neighbors.

The computation time and memory requirements for `'next'` and `'previous'` are the same as `'nearest'`.

Option for rounding numbers to a specified number of decimal or significant digits using the round function

The `round` function can now round to any specified number of decimal or significant digits.

For example, you can round `pi` to 2 decimal digits:

```
round(pi,2)
ans =

    3.1400
```

Or, you can round `pi` to 2 significant digits:

```
round(pi,2,'significant')
ans =

    3.1000
```

boundary function and alphaShape class for creating a conforming boundary around a discrete set of points

The `boundary` function and `alphaShape` class create enveloping boundaries, polygons, or polyhedra around a discrete set of 2-D or 3-D points, including options for tightening or loosening the boundaries around the points.

cummin and cummax functions for computing cumulative minimum and maximum of an array

The `cummax` and `cummin` functions compute the cumulative maximum and minimum values in an array. For example,

```
A = magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

B = cummin(A)

B =

    16     2     3    13
     5     2     3     8
     5     2     3     8
     4     2     3     1

C = cummax(A)

C =

    16     2     3    13
    16    11    10    13
    16    11    10    13
    16    14    15    13
```

Reverse accumulation option for the cumsum, cummin, cummax, and cumprod functions

The `'reverse'` option for `cumsum`, `cumprod`, `cummin`, and `cummax` reverses the direction of cumulation, working from `end` to `1` in the active dimension. This option allows quick directional calculations without requiring a flip or reflection of the input array.

Median and mode calculations of categorical data

The `median` and `mode` functions now support categorical arrays as inputs.

Functionality being removed or changed

Functionality	Result	Use Instead	Compatibility Considerations
`bitshift(A,k,N)`	Errors	`bitshift(A,k, assumedtype)`	Replace all instances of `bitshift(A,k,N)` with `bitshift(A,k,assumedtype)`.

Functionality	Result	Use Instead	Compatibility Considerations
`bitcmp(A,N)`	Errors	`bitcmp(A, assumedtype)`	Replace all instances of `bitcmp(A,N)` with `bitcmp(A,assumedtype)`.
`histc` function	Still Runs	`histcounts` function	Replace all instances of `histc` with `histcounts`

Data Import and Export

Faster data import from text files using Import Tool, and readtable and textscan functions

The performance of the Import Tool, the `readtable` function, and the `textscan` function improves for text files.

Import of data as categorical and datetime arrays using the readtable and textscan functions

The `readtable` and `textscan` functions can read data from text files a categorical or datetime arrays. Use the `%C` conversion specifier to read text as a category name. Use the `%D` conversion specifier to read text as a datetime value.

Data import from text files and collections of text files that do not fit into memory with datastore

The `datastore` function creates a datastore for reading collections of data that are too large to fit in memory. For example, `TabularTextDatastore` works with collections of text files.

VideoReader performance improvements and ability to start reading from a specified time in the video

The `VideoReader.hasFrame` and `VideoReader.readFrame` methods allow for improved performance when reading video files. The methods check for and read the next available frame in the video. You can set the `CurrentTime` property to begin reading from a specified time in the video file. This simplifies seeking into variable frame-rate video.

Compatibility Considerations

When you first create the `VideoReader` object, `VideoReader` no longer automatically populates the `NumberOfFrames` property. This behavior greatly improves performance, especially for large files. The `NumberOfFrames` property will be removed in a future release, and is now hidden when displaying the properties of a `VideoReader` object. When you use the `get` function to query a `VideoReader` object, the output structure array now has a `CurrentTime` field and no longer has a `Type` field or a `NumberOfFrames` field. You cannot query the `NumberOfFrames` property if you have invoked the `readFrame` or `hasFrame` methods, or if you have explicitly set the `CurrentTime` property.

If `VideoReader` cannot determine the duration of the video file, then the `Duration` property is empty (`[]`). Previously the `Duration` property was set to zero.

tcpclient function for reading and writing data from network connected devices and servers using socket-based connections

TCP, or Transmission Control Protocol, is a highly used networking protocol. The MATLAB TCP/IP client support uses raw socket communication and lets you connect to remote hosts in MATLAB for

reading and writing data. For example, you could use TCP/IP to acquire data from a remote weather station, and then plot the data.

You can create a TCP/IP connection to a server or hardware and perform read/write operations. Use the `tcpclient` function to create the connection, and the `write` and `read` functions for synchronously reading and writing data.

webread function for importing online data including JSON, CSV, and image data

The `webread` function reads content from RESTful Web services. You can use `webread` to import image, text, and JSON data from Web services into MATLAB. You also can use `websave` to write data imported from a Web service to file.

Reading non-ASCII encoded files with readtable function

The `readtable` function can read text files with non-ASCII character encoding schemes. Use the optional `'FileEncoding'` name-value pair argument to specify the encoding.

Writing quoted strings with writetable function

The `writetable` function can write quoted strings to text files. Use the optional `'QuoteStrings'` name-value pair argument to enclose MATLAB strings in double quotation marks when writing to a file.

hdftool functionality will not be removed

The R2013a Release Notes originally stated that `hdftool` would be removed in a future release. As of R2014b, there are no plans to remove this functionality. MATLAB no longer issues a warning when you call `hdftool`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`movie2avi`	Still Runs	`VideoWriter`	Remove all instances of `movie2avi`. Write to AVI files with `VideoWriter`.
`avifile`	Errors	`VideoWriter`	Replace all instances of `avifile` with `VideoWriter`.
`aviread`	Errors	`VideoReader`	Replace all instances of `aviread` with `VideoReader`.
`exifread`	Errors	`imfinfo`	Replace all instances of `exifread` with `imfinfo`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`urlread`	Still runs	`webread`	Use `webread` only for reading content with HTTP GET requests. To send data using HTTP POST, continue to use `urlread`.
`urlwrite`	Still runs	`websave`	Replace all instances of `urlwrite` with `websave`.
`read` method of `VideoReader` class	Still runs	`readFrame` method of `VideoReader` class	Replace all instances of `read` with `readFrame`.
`NumberOfFrames` property of `VideoReader` class	Still runs	none	Remove all instances of `NumberOfFrames`. When reading a video file, use the `CurrentTime` property to specify where reading should begin.
`hdfgd`	Still runs	https://www.mathworks.com/help/releases/R2014b/matlab/ref/matlab.io.hdfeos.gd.html `matlab.io.hdfeos.gd`	Replace all instances of `hdfgd` with the corresponding function in the `matlab.io.hdfeos.gd` package.
`hdfsd`	Still runs	https://www.mathworks.com/help/releases/R2014b/matlab/ref/matlab.io.hdf4.sd.html `matlab.io.hdf4.sd`	Replace all instances of `hdfsd` with the corresponding function in the `matlab.io.hdf4.sd` package.
`hdfsw`	Still runs	https://www.mathworks.com/help/releases/R2014b/matlab/ref/matlab.io.hdfeos.sw.html `matlab.io.hdfeos.sw`	Replace all instances of `hdfsw` with the corresponding function in the `matlab.io.hdfeos.sw` package.

Hardware Support

Documentation installation with hardware support package

Starting in R2014b, each hardware support package installs with its own documentation. See Supported Hardware for more information on support packages.

Support package for Android sensors

The MATLAB Support Package for Android Sensors allows you to collect sensor data from your mobile Android device and log it in MATLAB. You can then process the sensor data in MATLAB in a variety of ways, including creating plots. You can collect data from the following sensors:

- Acceleration
- Angular Velocity
- Orientation
- Magnetic Field
- Position

The Android Sensors product requires the following:

- MATLAB Mobile must be installed on your Android phone. You can acquire this app though the Google Play™ Store.
- MATLAB is required for creating the connection to the app and logging sensor data.
- Download and install the MATLAB Support Package for Android Sensors.

The support package includes the command-line interface in MATLAB and the ability to activate the Android Sensors mobile app, which is a separate tab in MATLAB Mobile. You must create a `mobiledev` object in MATLAB to connect to the sensors on the phone, and collect the data. The connection between your computer running MATLAB and the phone is done via Wi-Fi or your cellular network. To get started, see MATLAB Support Package for Android Sensors.

Support package for Arduino hardware

The MATLAB Support Package for Arduino Hardware allows you to send and receive data on Arduino devices. You can then process this data in MATLAB in a variety of ways, including creating plots. You can collect data from the following sensors connected to your Arduino hardware:

- I2C
- SPI
- Servo Motors

The support package includes the command-line interface in MATLAB. For more information see MATLAB Support Package for Arduino Hardware.

Support package for LEGO MINDSTORMS EV3 hardware

You can use MATLAB Support Package for LEGO® MINDSTORMS® EV3 Hardware on 32-bit and 64-bit Microsoft Windows systems, Mac OS, and Linux systems.

Use the LEGO MINDSTORMS EV3 hardware that connects to your system over a USB, Wi-Fi, or Bluetooth connection to:

- Get data from EV3 color, gyroscopic, infrared, ultrasonic, and touch sensors.
- Get data from an EV3 Remote Infrared Beacon.
- Control and get data from EV3 motors.
- Interact with the LCD, status light, speaker, and buttons on the EV3 brick.

Using the command-line interface in MATLAB, you can:

- Start programing without additional toolboxes.
- Interactively develop and debug programs in MATLAB.
- Acquire and process sensor data in MATLAB in a variety of ways, including creating plots.
- Run control loops at up to 25 Hz (not in real time).

See Install Support for LEGO MINDSTORMS EV3 Hardware.

For more information, see MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware.

Graphics

Major update of MATLAB graphics system

New MATLAB Graphics System

Starting in R2014b, the MATLAB graphics system is built on an improved infrastructure with a new look and includes many new features for improved charts and customizations. For more information on the new features and for compatibility considerations, see the remaining graphics release notes.

New look of MATLAB graphics with improved clarity and aesthetics

MATLAB graphics has a new look with many enhancements and new features, including:

- A new default colormap called `parula`, a lighter figure background color, and new colors when plotting multiple lines to emphasize plotted data.
- Anti-aliased fonts and lines for smoother text and graphics. For more information, see the `GraphicsSmoothing` property for figures and the `FontSmoothing` property for axes and text objects.
- New axes properties for setting the grid line colors and for controlling the title and axis label font sizes. For more information, see the `GridColor`, `TitleFontSizeMultiplier`, and `LabelFontSizeMultiplier` properties.
- Filled contour plot enhancements. For example, line breaks around the contour labels make the labels easier to read. Additionally, there are no longer outlines around the data limits, so they are easier to distinguish from contour lines.

Compatibility Considerations

Most of the new look changes are visual differences and should not affect your code. For troubleshooting topics related to the new look, see Why Are Plot Lines Different Colors? and How Do I Make the Graph Title Smaller?.

Improved infrastructure based on MATLAB objects

Starting in R2014b, the MATLAB graphics system is built on an improved infrastructure. Graphics objects now behave like other MATLAB objects:

- Graphics objects use object handles of various types instead of the numeric handles used in previous releases.
- Graphics objects support dot notation for getting and setting properties. Property names are case-sensitive when using dot notation.

Compatibility Considerations

Most code written for numeric handles still works with object handles. However, you should not perform operations that assume or require handles to be numeric values. For strategies to update existing code to work with object handles, see Graphics Handles Are Now Objects, Not Doubles.

Rotatable axis tick labels

New axes properties support rotated axis tick labels. For more information, see the `XTickLabelRotation`, `YTickLabelRotation`, and `ZTickLabelRotation` property descriptions.

Automatic update of datetime and duration tick labels with plot function

The `plot` function now supports the `datetime` and `duration` data types. Axis ticks and labels automatically update with the zoom, pan, and resize operations.

histogram function for plotting histograms

The `histogram` function plots histograms with data-dependent bin picking and options for bin control, normalization, and visualization.

Unlike the `hist` function, when you specify an output argument with the `histogram` function, it returns a histogram object that you can use to modify the properties of the histogram. For example, you can change the number of bins, use a different binning algorithm, or normalize the histogram in several ways. For more information, see Using histogram Objects.

animatedline function for creating line animations

The `animatedline` function optimizes line animations by accumulating and plotting data from a streaming data source.

Display of multilingual text and symbols

Axis tick labels support TeX and LaTeX markup for special characters, such as superscripts, subscripts, and Greek letters. By default, the axes interprets tick label characters using TeX markup. For more information, see the `TickLabelInterpreter` property of the axes. You also can use Unicode characters in axis tick labels, as well as in user interface objects.

Support for multiple colormaps in single figure

Figures support using a unique colormap for each axes in the figure. Specify an axes colormap by passing the axes handle to the `colormap` function.

Pie charts of categorical data with automatic slice labels

Pie charts support `categorical` data and automatically label the slices.

Image conversion functions rgb2gray and im2double, no longer requiring Image Processing Toolbox

MATLAB now includes the `rgb2gray` and `im2double` functions, which previously required the Image Processing Toolbox™. Use `rgb2gray` to convert RGB images or colormaps to grayscale. Use `im2double` to convert an image to double-precision values.

imshow function for displaying images from matrices or files, no longer requiring Image Processing Toolbox

MATLAB now includes the `imshow` function, which previously required the Image Processing Toolbox. Use `imshow` to display images from matrices or files.

savefig option that creates more compact files

Saving figures in `.fig` files with `savefig` creates more compact files using the `'compact'` option. Use this option with `.fig` files to be opened only in MATLAB version R2014b or later.

Compatibility considerations for graphics changes

The graphics changes introduced in R2014b support most of the functionality from previous releases, although there are some differences. Some of the changes that you are most likely to encounter are listed here.

- Plotting multiple lines with `hold on` uses the next color in the color order instead of starting from the first color with each plotting command.
- The `Clipping` property clips plotted objects to the six sides of the axes box defined by the axis limits. In previous releases, the `Clipping` property clips objects to the smallest 2-D rectangle that encloses the axes.
- If you add new data to a graph after using an `axis` command, such as `axis tight`, then MATLAB automatically updates the limits to incorporate the new data. In previous releases, MATLAB does not update the axis limits based on changes in the data.
- Colorbars and legends are no longer axes objects. They are new types of objects with their own sets of supported properties.
- The `Children` property for charting objects, legends, and colorbars no longer contains handles to underlying objects.
- `copyobj` no longer copies callbacks and application data. If you have existing code that copies object callback properties and object application data, use the `copyobj` function with the `'legacy'` option. This behavior is consistent with versions of `copyobj` before MATLAB release R2014b.

For troubleshooting topics related to these changes, see Graphics Changes in R2014b. For a list of removed properties and syntaxes, see the tables under “Properties and syntaxes being removed or changed” on page 13-20 and “Save and print functionality being removed or changed” on page 13-23.

Some graphics features might not work or might be unreliable because of outdated graphics drivers. For the best results with graphics, upgrade to the latest graphics drivers provided by your graphics hardware manufacturer. For more information, see System Requirements for Graphics.

Properties and syntaxes being removed or changed

Functionality	Result	Use Instead	Compatibility Considerations
`hold all` syntax	Still runs	`hold on`	Replace all instances of `hold all` with `hold on`.

Functionality	Result	Use Instead	Compatibility Considerations
`hist` function	Still runs	`histogram` function	Use `histogram` instead of `hist`.
`colorbar('peer',ax)` syntax	Still runs	`colorbar(ax)`	Replace all instances of `colorbar('peer',ax)` with `colorbar(ax)`.
Using `0` to access the root object	Still runs	`groot` command	Use the `groot` command to access the root object instead of `0`.
`EraseMode` property for all graphics objects	Warns	Not applicable	Remove all instances of the `EraseMode` property. For more information, see How Do I Replace the EraseMode Property?.
`noanimate` function	Warns	Not applicable	Remove all instances of the `noanimate` function. For more information, see How Do I Replace the EraseMode Property?.
`HitTestArea` property for group, transform, and chart objects	Errors	The `PickableParts` property	Remove all instances of the `HitTestArea` property. To control the area of an object that captures mouse clicks, use the `PickableParts` property.
Using the vertical slash character (|) in a string to separate text for tick labels or text objects	Does not separate text	Use a cell array of strings, a padded string matrix, or numeric vectors	Replace instances of using the vertical slash character (|) to define multiple strings with a cell array of strings, a padded string matrix, or numeric vectors.
Passing complex inputs to charting functions that do not support complex data	Warns or errors	Real valued input arguments	Charting functions that do not support complex data will warn or error if part of the data is ignored. Pass real valued inputs to these charting functions.
`'symbol'` option for `FontName` property	Warns	Use TeX or LaTex markup	Remove all instances of setting the `FontName` property to `'symbol'`. Use TeX or LaTeX markup instead. See the text `Interpreter` property for more information.
`'oblique'` option for `FontAngle` property	Uses italic instead	The `'italic'` option	Replace all instances of setting the `FontAngle` property to `'oblique'` with `'italic'`.
Using the `line` function with the dot value (`'.'`) for `LineStyle` property	Errors	The `':'` option for dotted lines	Replace all instances of setting the `LineStyle` property to `'.'` with `':'`.

Functionality	Result	Use Instead	Compatibility Considerations
`DrawMode` axes property	Warns	The `SortMethod` axes property	Remove all instances of the `DrawMode` property. Use the `SortMethod` property instead.
`NormalMode` property for surface and patch objects	Warns	The `VertexNormalsMode` or `FaceNormalsMode` property	Remove all instances of the `NormalMode` property. Use the `VertexNormalsMode` or `FaceNormalsMode` property instead.
`lighting phong` syntax	Uses gouraud lighting instead	`lighting gouraud`	Replace all instances of `lighting phong` with `lighting gouraud`.
`'phong'` value for the `EdgeLighting` and `FaceLighting` properties for surface and patch objects	Uses gouraud lighting instead	`'gouraud'` option	Replace all instances of setting the `EdgeColor` or `FaceColor` property to `'phong'` with the `'gouraud'` value instead.
Using the `Children` property for the axes to get the text objects for the title and axis labels	Does not return these objects	`Title`, `XLabel`, `YLabel`, and `ZLabel` axes properties	The text objects used for the title and axis labels are no longer children of axes. Use the `Title`, `XLabel`, `YLabel`, and `ZLabel` properties for the axes to get the text objects instead.
Passing text, image, rectangle, and annotation objects as an input argument to the `legend` function	Errors	Not applicable	Remove all instances of passing these objects to the `legend` function. Text, image, rectangle, and annotation objects are not included in the legend.
`neverselect`, `autoselect`, `advise`, `verbose` and `quiet` inputs for the `opengl` function	Warns	Not applicable	Remove all instances of using these inputs with the `opengl` function.
`'zbuffer'` option for the figure `Renderer` property	Uses OpenGL renderer instead	The `'opengl'` or `'painters'` options	Replace all instances of setting the `Renderer` property to `'zbuffer'` with `'opengl'` or `'painters'`. The default renderer is `'opengl'`.
`ResizeFcn` figure property	Still runs	`SizeChangedFcn` figure property	Use of the `ResizeFcn` figure property is not recommended. Use the `SizeChangedFcn` figure property instead.
`FixedColors`, `MinColorMap`, `WVisual`, `WVisualMode`, `XVisual`, and `XVisualMode` figure properties	Errors	Not applicable	Remove all instances of figure properties that are no longer supported. Use supported figure properties.
`Selected` and `SelectionHighlight` figure properties	No effect	Not applicable	Remove all instances of the `Selected` and `SelectionHighlight` figure properties.

Functionality	Result	Use Instead	Compatibility Considerations
`'fullcrosshair'` option for the figure `Pointer` property	Warns	Not applicable	Remove all instances of setting the figure `Pointer` property to `'fullcrosshair'`.
The root properties `CommandWindowSize`, `Diary`, `DiaryFile`, `Echo`, `Format`, `FormatSpacing`, `Language`, `More`, `RecursionLimit`, `BeingDeleted`, `ButtonDownFcn`, `UIContextMenu`, `Clipping`, `CreateFcn`, `DeleteFcn`, `BusyAction`, `Interruptible`, `HitTest`, `Selected`, `SelectionHighlight`, and `Visible`	Errors	Not applicable	Remove all instances of root properties that are no longer supported. Use supported root properties. For a list, see Root Properties.
Specifying an output argument for the `triplot` function	Returns single chart line object	Not applicable	Specifying an output argument for the `triplot` function returns a single chart line. Remove instances of code that rely on `triplot` returning multiple chart lines.
Specifying a single output argument for the `voronoi` function	Returns a vector of two chart line objects	Not applicable	Specifying a single output argument the `voronoi` function returns a vector of two chart lines. Remove instances of code that rely on `voronoi` returning more than two chart lines.

Save and print functionality being removed or changed

Functionality	Result	Use Instead	Compatibility Considerations
`pagesetupdlg` function	Warns	`printpreview`	Replace all instances of `pagesetupdlg` with `printpreview`.
`-crossplatform` and `-setup` options for `printdlg` function	Warns	Not applicable	Remove all instances of using the `-crossplatform` and `-setup` options with `printdlg`.
Using `getframe` function to capture content outside of the figure window	Warns	Specify area within figure window	Specify an area within a figure window when using `getframe` function.
`-dsetup` print option	Errors	Not applicable	Remove all instances of the `-dsetup` print option.
`'all'` option for `hgsave` and `hgload` functions	Errors	Not applicable	Remove all instances of using `hgsave` and `hgload` functions with the `'all'` option.

Functionality	Result	Use Instead	Compatibility Considerations
Setting the `PaperSize` figure property when the `PaperUnits` figure property is set to `'normalized'`	Errors	Not applicable	Remove instances of setting the `PaperSize` property when the `PaperUnits` property is set to `'normalized'`.
`mmat` option for `saveas` function	Errors	`savefig` function	Remove all instances of using the `mmat` option with `saveas`. To save a figure to a file, use `hgsave` with the `.fig` extension.
`printdmfile` function and `print -dmfile` syntax	Errors	`savefig` function or the figure menu	Remove all instances of `printdmfile` and `print -dmfile`. To save a figure to a file, use the `hgsave` function with the `.fig` extension, or use **File** > **Generate Code** on the figure menu instead.
`-dill` graphics format print option	Errors	`-depsc` or `-dsvg` print options for vector graphics formats	Remove all instances of using the `-dill` print option. For vector graphics formats, use `-depsc` or `-dsvg` option instead.
`-adobecset` print option	Errors	Not applicable	Remove all instances of the `-adobecset` print option.
The printer driver options `-dbj10e`, `-dbj200`, `-dbjc600`, `-dbjc800`, `-depson`, `-deps9high`, `-depsonc`, `-ddnj650c`, `-ddjet500`, `-dcdjmono`, `-dcdjcolor`, `-dcdj500`, `-dcdj550`, `-ddeskjet`, `-dlaserjet`, `-dljetplus`, `-dljet2p`, `-dljet3`, `-dljet4`, `-dpxlmono`, `-dpaintjet`, `-dpjxl`, `-dpjetxl`, `-dpjxl300`	Errors	Not applicable	Remove all instances of using these print options. To use a particular printer, install the printer driver.
The `-dps`, `-dpsc`, `-deps`, and `-depsc` print options generate PostScript Level 3 output instead of PostScript Level 1	PostScript Level 3 output	Not applicable	PostScript Level 1 output is no longer supported.

GUI Building

uitab and uitabgroup components for creating user interfaces with tabbed panels

The `uitab` function creates a tabbed panel in which you can group related components together with a clickable tab label. The `uitabgroup` function creates the container that manages user selection of uitabs.

Changes introduced with new graphics system

The new MATLAB graphics system introduces these changes to the GUI building tools:

- All UI components behave like MATLAB objects.
- Improvements to the stacking behavior of components for more consistent behavior.
- Improvements to component positioning for more consistency among different components and different platforms.

See Graphics Changes in R2014b for an overview of all the major changes.

Compatibility Considerations

- In some cases, the conditions and timing of the `ResizeFcn` callback execution have changed. See Why Has the Behavior of ResizeFcn Changed? for more information.
- The front-to-back order, or stacking order, of overlapping components might display differently in some UIs. See Why Are Some Components Missing or Partially Obscured? for more information.
- UI objects no longer support the use of `handle.listener` to create event listeners. See Why Does handle.listener Return an Error? for more information.
- The layout of UI components might change if you specify units in your code. Set the `Units` property before the `Position` property to ensure that MATLAB interprets the `Position` property values in those units. If you specify the `Units` property *after* the `Position` property, MATLAB interprets the `Position` values in the current units and converts the values to the equivalent values in the units you specified.
- Figure annotations and child objects of uipanels cannot cross uipanel boundaries. Previous versions of MATLAB allow annotations and child objects to extend into (or out of) the uipanel boundaries. Now, these items clip at the uipanel boundary. Annotations and child objects of uibuttongroups behave the same way when they encounter a uibuttongroup boundary.
- The `waitforbuttonpress` function is now figure-specific. The figure that is current when you call the `waitforbuttonpress` function is the only area in which users can press a key or click a mouse button to resume program execution. If you need to support multiple open figures, then use the `gcf` function to determine the current figure when you call `waitforbuttonpress`.

Functionality being removed or changed

Functionality	Result	Use Instead	Compatibility Considerations
`selectmoveresize`	No effect	`plotedit`	Replace all instances of `selectmoveresize` with `plotedit`.
The `Selected` and `SelectionHighlight` properties of `figure`, `uicontrol`, `uitable`, `uipanel`, `uibuttongroup`, `uipushtool`, and `uitoggletool`.	No effect	Not applicable	Remove all instances that use the `Selected` and `SelectionHighlight` properties. See “Properties and syntaxes being removed or changed” on page 13-20 for more information about all the changes to figure properties.
The `ResizeFcn` property of `figure`, `uipanel`, and `uibuttongroup`.	Still runs	`SizeChangedFcn` property	Use of the `ResizeFcn` property is not recommended. Use the `SizeChangedFcn` property instead. See “Properties and syntaxes being removed or changed” on page 13-20 for more information about all the changes to figure properties.
The `'fullcrosshair'` option for the figure `Pointer` property	Warning	Not applicable	Remove all instances of setting the figure `Pointer` property to `'fullcrosshair'`. See “Properties and syntaxes being removed or changed” on page 13-20 for more information about all the changes to figure properties.
The `SelectionChangeFcn` property of `uibuttongroup`	Still runs	`SelectionChangedFcn` property	Use of the `SelectionChangeFcn` property is not recommended. Use the `SelectionChangedFcn` property instead.
The `Clipping` property of `uicontrol`, `uitable`, and graphics `root`	Error	Not applicable	Remove all instances that use the `Clipping` property of a uicontrol, uitable, and graphics root. See “Properties and syntaxes being removed or changed” on page 13-20 for more information about all the changes to root properties.

Functionality	Result	Use Instead	Compatibility Considerations
The `'none'` option for the uipanel and uibuttongroup `BackgroundColor` property.	Error	Not applicable	Remove all instances of setting the uipanel or uibuttongroup `BackgroundColor` property to `'none'`.

Performance and Big Data

Big data analysis on your desktop that can scale to Hadoop with mapreduce

The `mapreduce` function enables analysis of data sets that do not fit in your computer's memory. It is used to process large data sets on your desktop, and can also be extended to run on Hadoop to process big data. MapReduce is a powerful technique for applying data processing methods to very large data sets, from simple statistics to complex machine learning algorithms.

For more information, including a selection of examples, see MapReduce.

The functionality of `mapreduce` extends beyond MATLAB with the following products:

- Access relational databases using Database Toolbox™
- Increased performance on desktops with Parallel Computing Toolbox
- Scaling up to Hadoop using MATLAB Distributed Computing Server™
- Create deployable archives or standalone applications that run against Hadoop using MATLAB Compiler

Improved performance for sorting categorical data with sort

The performance of the `sort` function improves for large categorical array inputs.

typecast function performance improvements with long vectors

The performance of the `typecast` function improves for long input vectors.

R2014a

Desktop

Pop-up Command History for recalling, viewing, filtering, and searching recently used commands in the Command Window

Pop-up Command History (2 min, 41 sec)

The Command History now displays in response to the up arrow (↑) in the Command Window, by default. Previously, the Command History window occupied a designated space on the MATLAB desktop. The following are additional features enhancing the Command History.

- Colored marks on the left side of the Command History indicate commands that generate errors. These marks are of the same color as error messages in the Command Window.
- You can perform case-insensitive searches in the Command History, as well as search for partial matches anywhere in a command.
- You can select multiple commands using **Shift** + ↑ and rerun them at once. In addition, brackets display on the left side of the Command History to indicate commands processed as a group. Select the bracket to rerun the group.
- You can filter commands to display only the results that match your search.
- The Command History can display the command execution time.

For more information, see Command History.

Compatibility Considerations

To view the Command History window in a docked location as in previous releases of MATLAB, click and then select **Dock**.

There is no longer any performance issue that justifies saving every *n*th command to the command history file. Therefore, this option no longer appears in the Command History Preferences. Instead, specify the total number of commands to save. The default is 25,000 commands, but you can specify a value up to 1,000,000. For more information, see Command History Preferences.

The history file is now named `History.xml`. Previously, it was named `history.m`. The first time you access an existing `history.m` file, MATLAB automatically converts it to `History.xml`, and you do not need to take further action.

Merge option in MATLAB Comparison Tool for resolving differences between text files

When comparing text files in the MATLAB Comparison Tool you can now merge changes from one file to the other. Merging changes can be useful when resolving conflicts between different versions of files.

For details, see Comparing Text Files.

Saving workspace variables and their values to a MATLAB script

MATLAB now provides the ability to save workspace variables to a MATLAB script. Once the script is saved, you can regenerate the workspace variables by running the script. Click **Save Workspace** on

the MATLAB desktop and select **MATLAB script (*.m)** in the **Save as type** menu. Alternately, use `matlab.io.saveVariablesToScript` to perform this operation from the command line.

Variables that cannot be saved to a script are saved to a MAT-file with the same name as that of the script.

Korean and Chinese localization available on Windows and Mac platforms

If the Language setting on your computer is Korean or Chinese, then MATLAB now has a localized user interface and documentation.

Korean and Chinese documentation is also available on the Web. For details, see Korean and Chinese Documentation.

MathWorks file properties displayed for .SLX, .SLXP, and .MLAPPINSTALL files in file browsers and search engines on Windows and Mac systems

Users can find and organize MathWorks files that are based on the Open Packaging Conventions format (OPC) directly through the Windows and Mac operating systems. MathWorks OPC files are files with `.slx`, `.slxp`, or `.mlappinstall` file extensions.

Use the `Tags` property to add custom searchable text to the file.

Language and Programming

Suggested corrections for mistyped, user-defined functions in the Command Window

MATLAB suggests corrections for function names mistyped in the command window. This functionality now includes suggestions for custom, or user-defined, functions on the MATLAB path.

Streamlined MEX compiler setup and improved troubleshooting

mex -setup is no longer necessary in most situations

Setting up `mex` compilers has been simplified. For most users, there is no longer any need to run `mex -setup`. `mex` automatically locates and uses a supported installed compiler.

mex reports selected compiler and success status

`mex`, by default, identifies the compiler it is using at build time and reports success. To suppress this feature, use the new `-silent` option. `mex` still reports errors and warnings, even when you specify `-silent`.

mex maintains different settings for C and C++

`mex` maintains different settings for C and C++ compilers.

Compiling for MATLAB engine applications is different

To build an engine application or a standalone application to read MAT-files, use the `mex` command with the new `-client engine` option. For more information, see What You Need to Build Engine Applications or What You Need to Build Custom Applications.

mex uses standard quoting and no escape characters

If you specify *`varname=varvalue`* parameters when you invoke `mex`, you no longer need to use double-quotes (`"`) on Windows, or escape the `$` character on Linux. To redefine a variable, use MATLAB-style single quotes (`'`) and no escape characters.

mex -setup takes a language setting

`mex -setup` selects a compiler for a given language, `lang`. If `lang` is not specified, `mex -setup` searches for C compilers.

Compatibility Considerations

- Do not use the `-f` option to build engine and MAT-file applications. Use the `-client engine` option instead.
- The format of the `mex` configuration files has changed. If there is a `.bat` or `.sh` options file in the current or `prefdir` folder, MATLAB displays a warning. In a future release, these warnings will become errors.
- Informational messages from `mex` have changed. The `mex` command displays the selected compiler and success status. To suppress these messages, use the `-silent` option.

- The output of the `mex -setup` command has changed. Use `mex -setup` to change the default compiler for a given language. For more information, see Changing Default Compiler.
- The `mex -setup` syntax has changed. You can specify a language. If none is specified, `mex -setup` uses `C`.

Multidimensional array support for flipud, fliplr, and rot90 functions

The `flipud`, `fliplr`, and `rot90` functions now support arrays with any number of dimensions. These functions operate independently on the planes formed by the first and second dimensions.

Option for circshift to operate on a specified dimension

The syntax `circshift(A,K,dim)` allows you to shift the elements of array `A` by `K` positions along dimension `dim`. For more information, see the reference page for `circshift`.

Compatibility Considerations

The behavior of `Y = circshift(A,K)` for scalar `K` will change in a future release. Currently, the function acts on the first dimension of `A` by default. In a future release, the function will operate on the first dimension of `A` whose size does not equal 1 by default. To retain current behavior, use `circshift(A,[K 0])`.

Changes to empty string matching with validatestring

`validatestring` now validates empty strings.

```
V = validatestring('',{'Cantor','','Koch'})
```

```
V =

     ''
```

In versions of MATLAB prior to R2014a, the call to `validatestring` results in an error.

Compatibility Considerations

Change code that relies expects an error when testing the validity of an empty string.

matlab.lang.makeValidName and matlab.lang.makeUniqueStrings functions for constructing unique MATLAB identifiers

The `matlab.lang.makeValidName` function returns valid MATLAB identifiers from input strings. The output names are not guaranteed to be unique. For example,

```
S = {'Item#','Item#','Price/Unit','1st order','Contact'};
N = matlab.lang.makeValidName(S)
```

```
N =

    'Item_'    'Item_'    'Price_Unit'    'x1stOrder'    'Contact'
```

The `matlab.lang.makeUniqueStrings` function returns unique strings from input strings. For example,

```
S = {'coffee' 'tea' 'coffee' 'water' 'coffee'};
U = matlab.lang.makeUniqueStrings(S)

U =

    'coffee'    'tea'    'coffee_1'    'water'    'coffee_2'
```

Use these two new functions together to ensure that output strings are valid and unique MATLAB identifiers. For example,

```
S = {'Item#','Item#','Price/Unit','1st order','Contact'};
N = matlab.lang.makeValidName(S);
U = matlab.lang.makeUniqueStrings(N,{},namelengthmax)

U =

    'Item_'    'Item__1'    'Price_Unit'    'x1stOrder'    'Contact'
```

details function displays details about arrays

For more information, see `details`.

Changes to passing empty object to isprop

Calls to `isprop` with empty objects now return an empty logical array instead of `false`.

Compatibility Considerations

Change code that relies on `isprop` retuning `false` for empty objects.

Behavior change of fullfile function output

In R2012b and earlier, the `fullfile` function returns a full file specification that includes repeated file separators and relative path symbols if the input file parts include them. For example, repeated file separators include `\\` and `\\\`. Relative path symbols include the dot (.) and double-dot (..) notation.

As of R2013a, `fullfile` collapses inner repeated file separators and relative directories indicated by the dot and double-dot symbols, and does not display those symbols in the output full file specification. `fullfile` maintains repeated file separators only if they appear at the beginning of the full file specification. `fullfile` maintains dot and double-dot symbols only if they appear at either end of the full file specification.

For example, the commands,

```
f = fullfile('C:\foo\folder1', '..\folder2')
g = fullfile('C:\foo\folder1', '\\\folder3\\')
h = fullfile('\\folder4\.')
```

return:

```
f =
C:\foo\folder2

g =
C:\foo\folder1\folder3\

h =
\\folder4\.
```

In R2012b and earlier, the same commands returned:

```
f =
C:\foo\folder1\..\folder2

g =
C:\foo\folder1\\\folder3\\

h =
\\folder4\.
```

Compatibility Considerations

Change code that relies on locating exact strings containing repeated file separators or dot and double-dot symbols in a full file path.

Support array-creation functions in your class

Class authors can add support to their classes for the array creation functions (`ones`, `zeros`, `rand`, `eye`, `NaN`, `inf`, `true`, `false`, `cast`, `rand`, `randn`, and `randi`). This support includes the class name and prototype object syntaxes. For more information, see Class Support for Array-Creation Functions.

Custom plugins in unit testing framework

The `matlab.unittest` testing framework provides an interface class, `matlab.unittest.plugins.TestRunnerPlugin`, to create custom plugins and extend the `TestRunner`. For more information, see Write Plugins to Extend TestRunner and Create Custom Plugin.

Test parameterization and selection in unit testing framework

Test authors can write tests that are parameterized to combine and execute over lists of data. For more information, see Create Basic Parameterized Test and Create Advanced Parameterized Test.

The `matlab.unittest.TestSuite.selectIf` method, combined with classes in the `matlab.unittest.selectors` package, allows for the improved selection of tests included in the test suite.

matlab.unittest plugin for Test Anything Protocol (TAP) output

The `matlab.unittest` testing framework provides a plugin that produces a Test Anything Protocol (TAP) stream. This plugin allows integration of MATLAB unit test results into third-party systems that recognize the Test Anything Protocol such as continuous integration systems. For more information, see the `matlab.unittest.plugins.TAPPlugin` documentation.

Output stream direction for matlab.unittest plugins

The `matlab.unittest` testing framework provides a means to redirect text output from several plugins to standard output (`ToStandardOutput`) or to a file (`ToFile`). Output stream direction is supported for the `DiagnosticsValidationPlugin`, `FailureDiagnosticsPlugin`, `TAPPlugin`, and `TestSuiteProgressPlugin` plugins.

Additionally, the `OutputStream` class provides an interface for test authors to create custom output streams.

Comparator for MATLAB objects in unit testing framework

The `PublicPropertyComparator` can be used with the `IsEqualTo` constraint to compare public properties of MATLAB objects recursively.

Changes to compiler support for building MEX-files

MATLAB supports Visual C++ 2013 compilers for building MEX-files on Microsoft Windows 32- and 64-bit platforms.

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

Changes to External Programming Language Interfaces documentation

External Programming Language Interfaces documentation is divided into two new categories in Advanced Software Development.

- Calling External Functions—How to use functionality from other languages, such as Java, .NET, and C, in MATLAB.
- MATLAB API for Other Languages—How to interact with MATLAB and MATLAB data types from other language applications. How to write and call MEX-functions.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`circshift(A,K)` for scalar `K`.	Still runs	`circshift(A,[K 0])` to retain current behavior	In a future release, the behavior of `circshift(A,K)`, where `K` is a scalar, will change. The function will operate on the first array dimension whose size does not equal 1 by default. Currently, the function operates on the first array dimension by default. To retain current behavior, use `circshift(A,[K 0])`.
`flipdim(A,dim)`	Still runs	`flip`	Replace all instances of `flipdim(A,dim)` with `flip(A)` or `flip(A,dim)`.
`bitmax`	Warns	`flintmax`	Replace all instances of `bitmax` with `flintmax`.
`tstool`	Errors	`timeseries`, `tscollection`, `openvar`, or `plot`	Replace all instances of `tstool` with `timeseries`, `tscollection`, `openvar`, or `plot`. To create a time series object, use `timeseries`. To create a time series collection with one or more `timeseries` objects, use `tscollection`. To open a time series object or collection in the Variables editor, use `openvar`. To plot a time series object, use `plot`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations

`genvarname`	Still runs	`matlab.lang.makeValidName` and `matlab.lang.makeUniqueStrings`	Replace all instances of `genvarname` with `matlab.lang.makeValidName` and `matlab.lang.makeUniqueStrings`. For example,
			`S = {'my.Name','my_Name','my_Na`
			`N = matlab.lang.makeValidName(S`
			`U = matlab.lang.makeUniqueStrin`
			`{},namelengthmax);`

`depdir`	Warns	`matlab.codetools.requiredFilesAndProducts`	Replace all instances of `depdir` with `matlab.codetools.requiredFilesAndProducts`. However, `matlab.codetools.requiredFilesAndProducts` returns the full path of a required file, including the file name.
`depfun`	Warns	`matlab.codetools.requiredFilesAndProducts`	Replace all instances of `depfun` with `matlab.codetools.requiredFilesAndProducts`. However, `matlab.codetools.requiredFilesAndProducts` does not identify opaque classes, such as Java or COM classes. There is no replacement for this functionality.
`createCopy` method of `inputParser` class	Errors	`copy`	Replace all instances of `createCopy` with `copy`.

`mex -f` *`filepath`*`\`*`mexopts`*`.bat`
 	
`mex -f` *`filepath`*`/`*`mexopts`*`.sh`
	Warns	For building engine and MAT-file applications, use `mex -client engine`.
		
		Replace custom `mex` options files with `mex` command-line options.	

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`mexIsGlobal` in the C/C++ and Fortran MEX API Library	Always returns `false`	`mxIsFromGlobalWS` in the C/C++ Matrix Library API and Fortran Matrix Library API	Replace all instances of `mexIsGlobal` with `mxIsFromGlobalWS` in MEX-files.

Mathematics

isdiag, isbanded, issymmetric, ishermitian, istril, istriu, and bandwidth functions for testing matrix structure

The following functions test various aspects of matrix structure and are useful in simplifying numerical algorithms.

- `ishermitian` determines if a matrix is Hermitian or skew-Hermitian.
- `issymmetric` determines if a matrix is symmetric or skew-symmetric.
- `istriu` determines if a matrix is upper-triangular.
- `istril` determines if a matrix is lower-triangular.
- `isdiag` determines if a matrix is diagonal.
- `bandwidth` returns the upper and lower bandwidth of a matrix.
- `isbanded` determines if a matrix is within the specified upper and lower bandwidths.

sylvester function for solving the Sylvester equation

The `sylvester` function solves the Sylvester equation, $AX + XB = C$, for `X`.

Option for eig function for computing left eigenvectors

The `eig` function can now calculate the left eigenvectors of a matrix `A` explicitly.

For example:

```
A = magic(3);
[V,D,W] = eig(A)

V =

   -0.5774   -0.8131   -0.3416
   -0.5774    0.4714   -0.4714
   -0.5774    0.3416    0.8131


D =

   15.0000         0         0
         0    4.8990         0
         0         0   -4.8990


W =

   -0.5774   -0.7416   -0.0749
   -0.5774    0.6667   -0.6667
   -0.5774    0.0749    0.7416
```

This produces a full matrix, `W`, whose columns are the left eigenvectors of `A` such that `W'*A = D*W'`.

Option for rand, randi, and randn functions for creating arrays of random numbers that match data type of an existing variable

The functions `rand`, `randi`, and `randn` can now return output that matches the data type of an existing variable.

For example:

```
A = int16(32);
r = randi(A,4,4,'like',A);
class(r)

ans =

int16
```

Integer type support for mean

The `mean` function now supports inputs of any integer data type.

complex function with one complex input

The `complex` function now supports one complex input. If `X` is complex, then `z = complex(X)` is identical to `X`. In previous releases, `complex` returned an error.

Change to ind2sub and sub2ind functions with nondouble inputs

The output behavior of the `ind2sub` and `sub2ind` functions has changed. The new output of these functions always has class `double` regardless of the class of the input.

Compatibility Considerations

In previous releases, the output class of these functions was dependent on the input class. To obtain nondouble output, cast the output into the required class, such as `int8(sub2ind(size,i,j))`.

Data Import and Export

Webcam support for previewing and acquiring live images and video

Webcam Support (2 min, 54 sec)

You can use the MATLAB Webcam support to bring live images from any USB Video Class (UVC) Webcam into MATLAB. This includes Webcams that might be built into laptops or other devices, as well as Webcams that plug into your computer via a USB port.

With simple MATLAB functions you can detect your connected Webcams, acquire single snapshots from a Webcam, and optionally set up a loop of acquired images. The `webcamlist` function allows you to detect the connected Webcams. The `webcam` function creates the Webcam object that is used to acquire images. And the `snapshot` function returns a single image from the camera. You can also preview your image and set properties for the image.

The Webcam support is available only through the Hardware Support Packages. You must download and install the necessary files using the Support Package Installer. To open the Support Package Installer, type `supportPackageInstaller` in MATLAB. Then on the **Select support package to install** screen, select the `USB Webcams` from the list. For more information on installing this support package, see Installing the Webcam Support Package.

The MATLAB Webcam support can be used on the following platforms:

- Microsoft Windows 32-bit and 64-bit
- Mac OS X 64-bit
- Linux

For more information about using the Webcam feature, see

- Connecting to Webcams — how to use the `webcamlist` function to detect your cameras
- Acquiring Images from Webcams — how to acquire live images from your camera into MATLAB
- Setting Properties for Webcam Acquisition — how to set object- or device-specific properties for the acquisition

Raspberry Pi hardware support for controlling devices such as motors and actuators, and for capturing live data from sensors and cameras directly from MATLAB

You can use MATLAB commands to connect to a Raspberry Pi board over a network and perform the following operations:

- Exchange data with sensors and actuators that are connected to the GPIO, serial port, I2C, and SPI interfaces
- Record video and take photographs using the Camera Board
- Issue Linux shell commands
- Transfer files to or from your host computer
- Control the on-board LED

To install or update this support package, perform the steps described in Install Support for Raspberry Pi Hardware.

For more information, see Raspberry Pi Hardware.

readtable improvements for reading spreadsheet and text files

- The `readtable` function now automatically recognizes `.xlsb`, `.xlsm`, `.xltm`, `.xltx`, and `.ods` as file extensions for spreadsheet files. You no longer need to specify the `'FileType','spreadsheet'` name-value pair argument when reading files with these extensions.
- The `readtable` function can now read from text files without a file extension. Previously, `readtable` searched for a file with a `.txt` extension.
- On systems with Excel for Windows, the `readtable` function can read spreadsheet files in basic mode, using the `'Basic'` name-value pair argument. Basic mode is the default for systems without Excel for Windows.

Compatibility Considerations

If you specify a file name with no extension, for example, `foo`, the `readtable` function reads `foo` as a text file, if it exists. If `foo` does not exist, then `readtable` searches for and reads from `foo.txt`. In R2013b, `readtable` reads only from the file named `foo.txt`.

To read from a file with a `.txt` extension when an identically named file without an extension also exists, specify both the file name and extension in the call to `readtable`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`aufinfo`	Warns	`audioinfo`	Replace all instances of `aufinfo` with `audioinfo`.
`auread`	Warns	`audioread`	Replace all instances of `auread` with `audioread`.
`auwrite`	Warns		Write audio files using `audiowrite`.
`wavfinfo`	Warns	`audioinfo`	Replace all instances of `wavfinfo` with `audioinfo`.
`wavread`	Warns	`audioread`	Replace all instances of `wavread` with `audioread`.
`wavwrite`	Warns	`audiowrite`	Replace all instances of `wavwrite` with `audiowrite`.
`wavplay`	Errors	`audioplayer` and `play`	Replace all existing instances of `wavplay` with `audioplayer` and `play`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`wavrecord`	Errors	`audiorecorder` and `record`	Replace all existing instances of `wavrecord` with `audiorecorder` and `record`.
`mmreader`	Errors	`VideoReader`	Replace all instances of `mmreader` with `VideoReader`.

GUI Building

Panel Display in GUIDE Layout Area

While designing a GUI with GUIDE, if you place a panel on top of controls, the layout remains as you specify in the layout area. However, in the running GUI, the panel displays under the controls.

Compatibility Considerations

Previously, if you placed a panel on top of controls in the layout area, the panel automatically moved under the controls in the design area and appeared under the controls in the running GUI. Now, the panel remains as you place it in the layout area, but displays under the controls in the running GUI. The best practice is to place the panel in the layout area first, and then place the controls in the panel. This way, the design area matches the appearance of the running GUI. For existing GUIs, you can right-click the panel in the design area, and then select **Send to Back**.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`HitTest`, `Selected`, and `SelectionHighlight` properties for these components: • figure • uicontrol • uipanel • uibuttongroup • uipushtool • uitable • uitoggletool • uitoolbar	Still runs	Not applicable	Use of the `HitTest`, `Selected`, and `SelectionHighlight` properties is not recommended for the listed components. These properties might be removed from these components in a future release.

Performance

conv2 function performance improvements with three inputs

The performance of the `conv2` function improves when using the `conv2(h1,h2,A)` syntax.

filter function performance improvements for FIR and IIR

The performance of the `filter` function, `filter(b,a,X)`, improves in the following cases:

- where `a` is a scalar and `X` is a sufficiently long vector (FIR filter)
- where `a` is a vector and `X` is a vector, matrix, or multidimensional array (IIR filter)

R2013b

Language and Programming

table data container for managing, sorting, and filtering mixed-type tabular data

Tables and Categorical Arrays (6 min, 9 sec)

`table` is a new data type to collect mixed-type data and metadata properties, such as variable names, row names, descriptions, and variable units, in a single container. Tables are suitable for column-oriented or tabular data that is often stored as columns in a text file or in a spreadsheet. For example, you can use a table to store experimental data, with rows representing different observations and columns representing different measured variables.

Tables consist of rows and column-oriented variables. Each variable (column) in a table can have a different data type and a different size with the restriction that each variable must have the same number of rows. For example,

```
T =

               Gender    Age    Smoker     BloodPressure
               ------    ---    ------    ---------------
    Smith      'M'       38     true      124          93
    Johnson    'M'       43     false     109          77
    Williams   'F'       38     false     125          83
    Jones      'F'       40     false     117          75
    Brown      'F'       49     false     122          80
```

For more information, see Tables.

categorical array for ordered and unordered categorical data

Tables and Categorical Arrays (6 min, 9 sec)

`categorical` is a data type to store data with values from a finite set of discrete categories. A categorical array provides efficient storage and convenient manipulation of nonnumeric data, while also maintaining meaningful names for the values. Ordinal categorical arrays are a type of categorical array whose categories have a mathematical order. For example,

```
myCategorical =

     medium      large      small      small      medium

categories(myCategorical)

ans =

    'small'
    'medium'
    'large'
```

You can use categorical arrays in a table to select groups of rows. For more information, see Categorical Arrays.

timeit function for robust time estimates of function execution

The `timeit` function measures the time required to run a function. It provides a more robustly computed time estimate than `tic/toc`.

localfunctions function for getting handles to all local functions in a file

The `localfunctions` function returns a cell array of function handles to all local functions in the current file.

Functions for writing, executing, and verifying tests using the matlab.unittest testing framework without creating custom classes

As an alternative to writing object-oriented tests, the MATLAB xUnit-style testing framework now provides function-based writing, execution, and verification of tests. For more information, see Unit Testing Framework. For an example of function–based test writing, see Write Simple Test Case Using Functions.

matlab.mixin.CustomDisplay utility class to write custom display methods

Use the `matlab.mixin.CustomDisplay` class to customize object display for your MATLAB class.

flip function, a faster and more memory efficient alternative to flipdim for flipping arrays and vectors

The `flip` function provides a faster and more memory efficient alternative to `flipdim` for flipping arrays and vectors.

Partial name matching in inputParser

The `inputParser` now includes the `PartialMatching` property, which allows partial matching of parameter names. This property is `true` by default. For more information, see the `inputParser` reference page.

Compatibility Considerations

Parsing schemes requiring exact parameter name matching should set the `inputParser PartialMatching` property to `false`.

Additional validateattributes options for checking array values

The `validateattributes` function now checks the following additional attributes of numeric or logical input arrays.

Attribute	Description
`'decreasing'`	Each column element is less than the previous element and no element is `NaN`.
`'increasing'`	Each column element is greater than the previous element and no element is `NaN`.
`'nondecreasing'`	Each column element is greater than or equal to the previous element and no element is `NaN`.
`'nonincreasing'`	Each column element is less than or equal to the previous element and no element is `NaN`.

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

Conversion changes of out-of-range numbers passed to Java methods that take integers

MATLAB R2013b changes the way it converts out-of-range values for the following Java integer types:

- Signed 32-bit integer (Java `int`, `short` or `byte` parameters)
- Signed 64-bit integer (Java `long` parameters)

For a description of the conversion, see Converting Numbers to Integer Arguments.

Compatibility Considerations

If your MATLAB code can pass out-of-range values to Java methods with integer type arguments, the results might change. For example, the following value, `val`, is out-of-range for the argument to `java.lang.Integer`.

```
val = uint32(2^31);
java.lang.Integer(uint32(val))
```

In previous versions of MATLAB, the result is:

```
ans =

-2147483648
```

As of MATLAB R2013b, the result is:

```
ans =

-1
```

Additional properties for mex.getCompilerConfigurations function

The `mex.getCompilerConfigurations` function returns the following new properties:

- `ShortName` — Character string used to identify options file for the compiler
- `MexOpt` — Name and full path to options file
- `Priority` — The priority of this compiler

Changes to compiler support for building MEX-files

MATLAB no longer supports the following compilers on Microsoft Windows platforms:

- Intel C++ Version 11
- Intel Visual Fortran Version 11
- Open Watcom C/C++

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

Changes to time alignment for time series objects

The `IsTimeFirst` property for time series objects is now read only. The property value is `false` for 3-D and higher dimensional data. Otherwise, the value is `true`.

- `true` — The first dimension of the data array is aligned with the time vector. For example, `ts = timeseries(rand(3,3),1:3);`
- `false` — The last dimension of the data array is aligned with the time vector. For example: `ts = timeseries(rand(3,4,5),1:5);`

Consequently, the alignment of the `Time` property with the first or last data dimension of the `Data` property is based on the number of dimensions of the data. When the data contains three or more dimensions, the length of the `Time` property matches the size of the last data dimension. Otherwise, the length of the `Time` property matches the size of the first data dimension.

Compatibility Considerations

You receive an error when creating a time series object with 3-D or higher dimensional data and with time alignment along the first dimension.

To preserve the property value of `true` for `IsTimeFirst` with 3-D or higher dimensional data, reshape the data array to two dimensions, such that the last data dimension and the time dimension are compatible. For example, replace `timeseries(ones(10,4,2),1:10)` with `timeseries(ones(10,8),1:10)`.

To preserve the number of dimensions of a 3-D or higher dimensional data array, use `permute` to reorder the data array, such that the size of the last data dimension aligns with the time vector.

Furthermore, you receive a warning when loading a time series object from a prior release with 3-D or higher dimensional data and with time alignment along the first dimension. In this case, MATLAB preserves the time alignment with the first dimension and reshapes the data to 2-D.

New fixture and plugin features for matlab.unittest testing framework

The `matlab.unittest` testing framework now provides four customized fixtures to ease the creation of setup and teardown code. You can use these fixtures to change the current working folder, add a folder to the MATLAB path, suppress the display of warnings, and create a temporary folder. For more information, see `matlab.unittest.fixtures`.

To share these fixtures across test classes, use the new `SharedTestFixtures` class attribute of `TestCase`. The `getSharedTestFixtures` method of `TestCase` provides access to the shared fixtures. You also can use fixtures within a test function by calling the `applyFixture` method of `TestCase`.

To pause execution of a test and enter debug mode upon a failure or uncaught error, you can add the new plugin, `StopOnFailuresPlugin` to the test runner. For more information, see `matlab.unittest.plugins`.

Conversion of error and warning message identifiers

For R2013b, error and warning message identifiers have changed in MATLAB.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `MATLAB:InputParser:MustBeChar` identifier has changed to display either `MATLAB:InputParser:MustBeChar` or `MATLAB:InputParser:ParamMustBeChar`. If your code checks for `MATLAB:InputParser:MustBeChar`, you might need to update it to check for `MATLAB:InputParser:ParamMustBeChar` instead. For a mapping of the new identifiers to the original identifiers, see Technical Support solution `1-ERAFNC`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`addParamValue` method of `inputParser` class	Still Runs	`addParameter`	Replace all instances of `addParamValue` with `addParameter`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`tstool`	Warns	`timeseries`, `tscollection`, `openvar`, or `plot`	Replace all instances of `tstool` with `timeseries`, `tscollection`, `openvar`, or `plot`. To create a time series object, use `timeseries`. To create a time series collection with one or more `timeseries` objects, use `tscollection`. To open a time series object or collection in the Variables editor, use `openvar`. To plot a time series object, use `plot`.
Time series object with 3-D or higher dimensional data and time aligned with the first dimension	Errors	Time series object with 2-D data and time aligned with the first dimension or a time series object with 3-D or higher dimensional data and time aligned with the last dimension	For more information about updating your code and assessing the impact of this change, see “Changes to time alignment for time series objects” on page 15-5.
`cat(0,A,B)`	Errors	`cat(1,A,B)`	In previous releases, the `cat` function silently changed `dim=0` to `dim=1`. Now, `cat` returns an error when `dim=0`. Replace all instances of `cat(0,A,B)` to `cat(1,A,B)`.

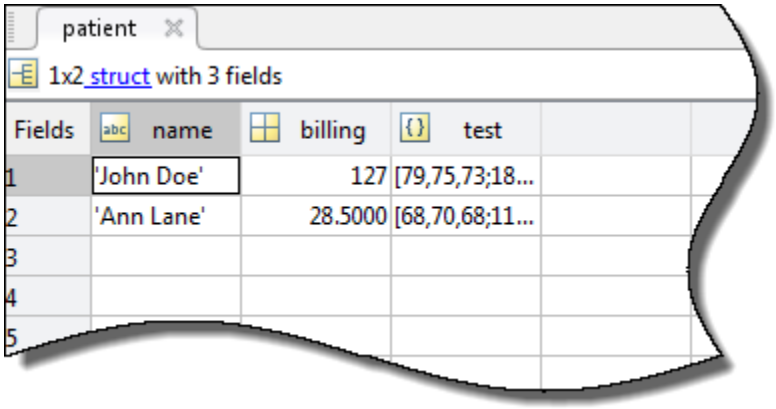
Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`cat(z,A,B)`, where `z` is a complex number	Errors	`cat(real(z),A,B)`	In previous releases, the `cat` function silently used the real part of `dim`. Now, cat returns an error when `dim` is complex. Replace all instances of `cat(z,A,B)` to `cat(real(z),A,B)`.
`mexIsGlobal` in the C/C++ and Fortran MEX API Library	Still Runs	`mxIsFromGlobalWS` in the C/C++ Matrix Library and Fortran Matrix Library	Replace all instances of `mexIsGlobal` with `mxIsFromGlobalWS` in MEX-files.

Desktop

Improved viewing and editing of one-dimensional structure arrays in the Variables editor

One-dimensional (n-by-1 or 1-by-n) structure arrays in the Variables editor now display field contents arranged in columns, in a single pane. You can reorder fields by dragging a column.

The new display allows you to work with one-dimensional structure arrays as you would with other workspace variables. For example, you can edit field values in-place, create new variables from a selection, and plot selected data using the options on the **Plots** tab.



For more information about editing structure arrays using the Variables editor, see Edit Table and Structure Array Data Interactively

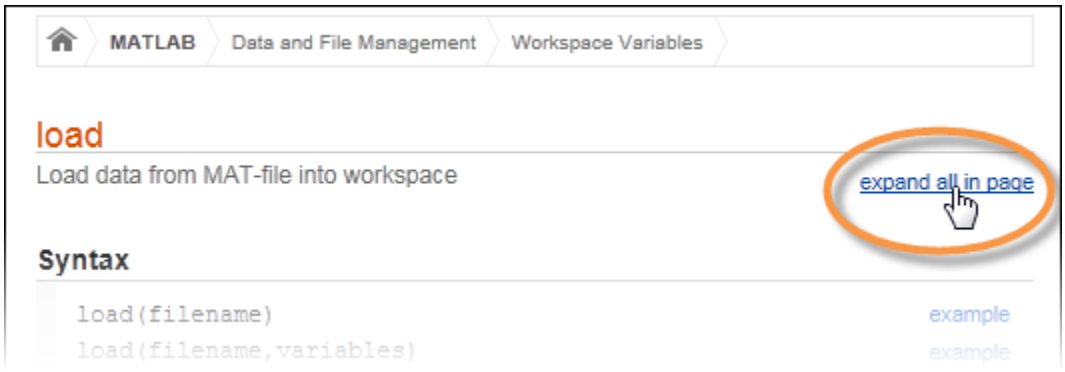
Improved management of a large number of open files, figures, and documentation pages

Files, docked figures, and documents each display in separate tabs that you can rearrange. For example, each file displays on a tab in the MATLAB Editor.

A drop-down list provides quick access to tabs that do not fit in the window when a large number of files, figures, or documents are open. When viewing multiple documents in a tiled layout, you can drag tabs to create new tiles or move documents between existing tiles. The tabs replace the document bar that was shared across tiles in earlier versions of MATLAB.

Expand all option for opening collapsed sections in documentation pages for printing and in-page searching

Some documentation pages include sections that are collapsed by default, such as input argument descriptions or examples. In-page searches do not find terms in those sections unless you first expand the sections. In R2013b, you can easily expand all sections on a page by clicking **expand all in page**, which is located at the top right of pages with collapsed sections. Then, you can search for terms anywhere on the page, or print the page in its entirety.



Java integration updated to version 7, providing access to new Java features and bug fixes

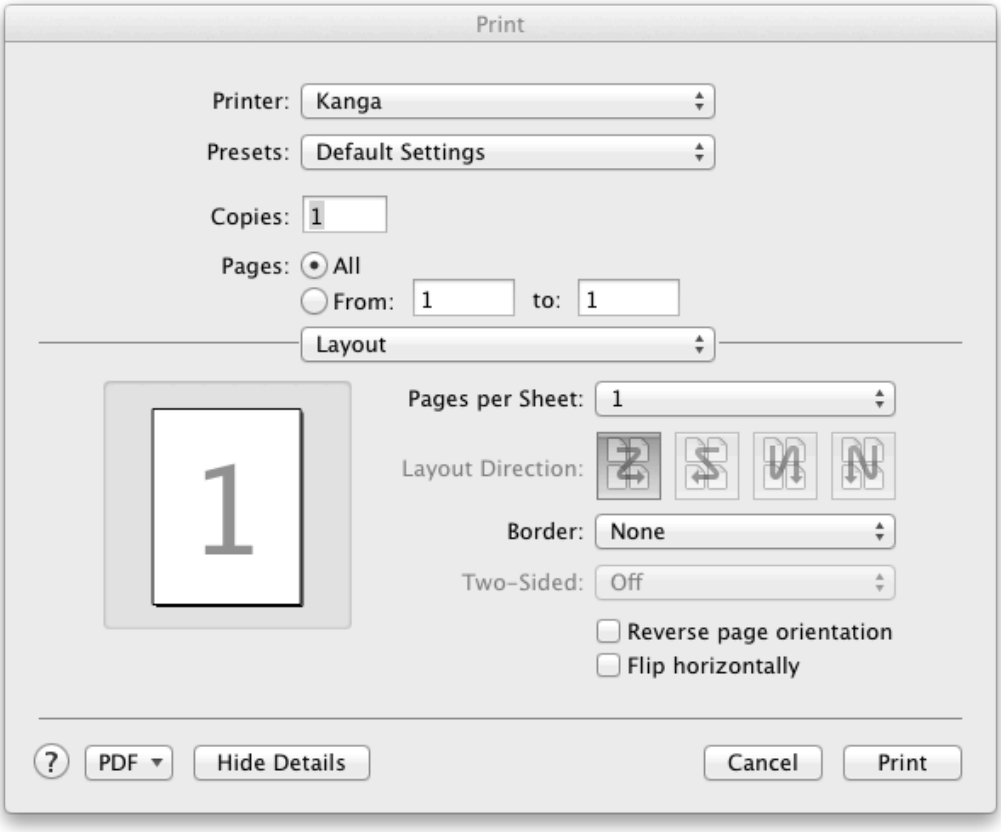
As of R2013b, support for the Oracle Java Runtime Environment (JRE) has been updated to Java 7 Update 11 on all platforms.

Bundling of Java on Mac, removing dependency on Apple supplied Java runtime

The Mac version of MATLAB is no longer dependent on the Apple-provided JRE.

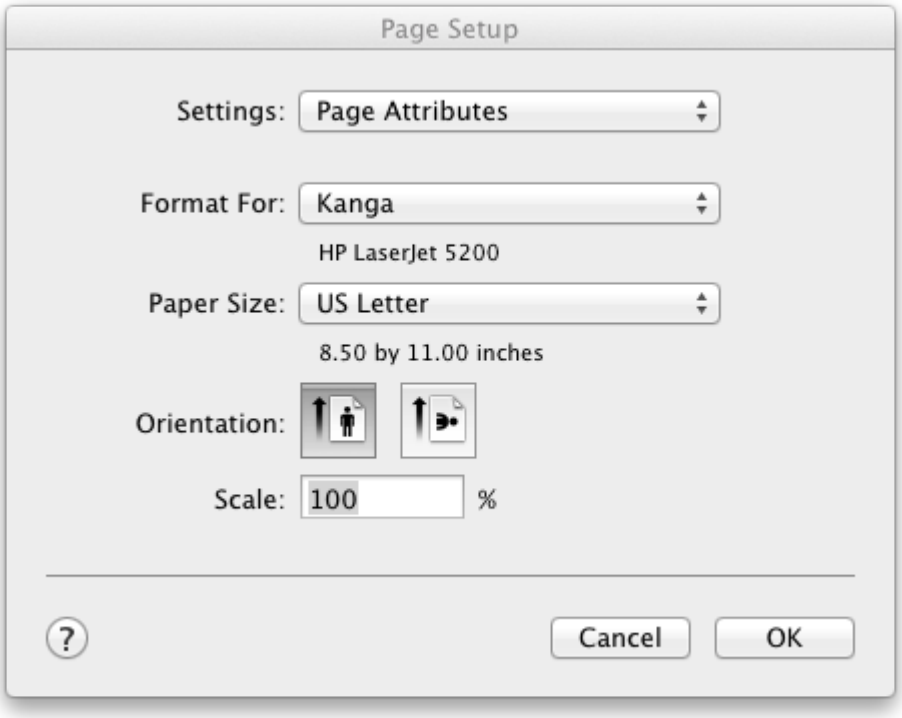
Enhanced print options on Mac operating systems

On Macintosh systems, the Print dialog box now provides more print options. You can access all print options provided by the native Macintosh print dialog. For example, you can select the number of pages to print per sheet, add a border or watermark, and print to a PDF file. To access the Print options, press **Command+P** from the Command Window or the MATLAB Editor.



Compatibility Considerations

Page Setup options, such as paper size and orientation, are no longer accessible from the Print dialog box. To access Page Setup options, select the **Editor** tab. In the **File** section, click **Print** ▾ and then select **Page Setup**. Alternatively, from the Command Window or Editor, use the keyboard shortcut, **Command**+**Shift**+**P**. The Page Setup dialog box opens. In the **Settings** menu, ensure that **Page Attributes** is selected.



To access the options previously available from the Page Setup dialog box (such as Layout, Header, and Fonts options), select **MATLAB** in the **Settings** menu.

Option for following documentation links to uninstalled products

By default, the Help browser displays only the documentation associated with your installed products. In previous releases, if you used this default and clicked a link to documentation in an uninstalled product, the Help system displayed a “Page Not Found” error. Now, the Help system asks if you want to view the page from the MathWorks Web site. If so, the page opens in the Help browser.

For information on changing the default settings, see Help Preferences.

Preferences dialog box improvements for easier navigation

Options in the left pane of the Preferences dialog box are arranged by product, and alphabetized within each product for simpler navigation. This change does not affect code that calls preferences. To access the Preferences dialog box, click **Preferences** on the **Home** tab.

Auto-adjust capability in Variables editor

You now can auto-adjust the column width for elements of numeric, cell, structure, and table arrays in the Variables editor. Point to the border to the right of a column heading, until a double-headed arrow appears. Then, double-click to auto-adjust the width of that column.

MATLAB support added to Windows 7 Default Programs control panel

You can manage MATLAB file associations on Microsoft Windows 7 systems using the **Set your default programs** option in the **Default Programs** control panel.

MATLAB no longer supports selective installation of individual file associations.

For more information, see Associate Files with MATLAB on Windows Platforms.

Japanese localization available on Mac platforms

If your Mac Language setting is Japanese, then MATLAB, Simulink, and other localized MathWorks products now have a Japanese user interface and documentation.

Mathematics

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`interp1(X, Y, Xq, [], ...)` where `[]` specifies the default interpolation method	Still Runs	`interp1(X, Y, Xq, 'linear', ...)`	Replace all instances of `interp1(X, Y, Xq, [], ...)` with `interp1(X, Y, Xq, 'linear', ...)`.

Passing mixed-orientation vectors to `interp2`:
`Vq = interp2(x, y, V, xq, yq)`
Specifically, if one or both of the following are true:
• One of `x` and `y` is a row vector and the other is a column vector.
• One of `xq` and `yq` is a row vector and the other is a column vector.
	Still Runs	Construct the full grid with `meshgrid` first. Alternatively, use `griddedInterpolant` if you have a large data set.	Modify all instances that pass mixed-orientation vectors to `interp2`. You can modify your code in one of two ways:
			• Call `meshgrid` to construct the full grid first.
			`[X,Y] = meshgrid(x, y);`
			`[Xq,Yq] = meshgrid(xq, yq);`
			`Vq = interp2(X, Y, V, Xq, Yq);`
			• Pass the vectors to `griddedInterpolant` inside a cell array.
			`F = griddedInterpolant({x, y}, V.');`
			`Vq = (F({xq, yq})).'`

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed-orientation vectors to `interp3`:
 	
`interp3(x, y, z, V, xq, yq, zq)` Specifically, if one or both of the following are true:
 	
• `x`, `y`, and `z` are a combination of row and column vectors.
• `xq`, `yq`, and `zq` are a combination of row and column vectors.
	Still Runs	Construct the full grid with `meshgrid` first. Alternatively, use `griddedInterpolant` if you have a large data set.	Modify all instances that pass mixed-orientation vectors to `interp3`. You can modify your code in one of two ways:
			
			• Call `meshgrid` to construct the full grid first.
			`[X,Y,Z] = meshgrid(x, y, z);`
			`[Xq,Yq,Zq] = meshgrid(xq, yq, zq);`
			`Vq = interp3(X, Y, Z, V, Xq, Yq, Zq);`
			• Pass the vectors to `griddedInterpolant` inside a cell array.
			`V = permute(V, [2 1 3]);`
			`F = griddedInterpolant({x, y, z}, V);`
			`Vq = F({xq, yq, zq});`
			`Vq = permute(Vq, [2 1 3]);`

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed-orientation vectors to `interpn`:
`interpn(x1,x2,...,xn, V, x1q,x2q,...,xnq)`
Specifically, if one or both of the following are true:
• `x1,x2,...,xn` are a combination of row and column vectors.
• `x1q,x2q,...,xnq` are a combination of row and column vectors.
	Still Runs	Construct the full grid with `ndgrid` first. Alternatively, use `griddedInterpolant` if you have a large data set.	Modify all instances that pass mixed-orientation vectors to `interpn`. You can modify your code in one of two ways:
			• Call `ndgrid` to construct the full grid first.
			`[X1,X2,...,Xn] = ndgrid(x1, x2, ..., xn); [X1q,X2q,...,Xnq] = ndgrid(x1q, x2q, ..., xnq); Vq = interpn(X1, X2, ..., Xn, V, X1q, X2q, ..., Xnq);`
			• Pass the vectors to `griddedInterpolant` inside a cell array.
			`F= griddedInterpolant({x1, x2, ..., xn}, V); Vq= F({x1q, x2q, ..., xnq});`

`interp1(..., 'cubic')`	Warns	`interp1(..., 'pchip')`	Replace all instances of `interp1(..., 'cubic')` with `interp1(..., 'pchip')`
Passing the `'pp'` flag to `interp1`. For example: `pp = interp1(x, v, 'linear', 'pp'); vq = ppval(pp, xq);`	Warns	Use `griddedInterpolant` to create an interpolating function that is efficient to evaluate in a repeated manner.	Replace all instances of `interp1(..., 'pp')` with `griddedInterpolant`. For example: `F = griddedInterpolant(x, v, 'linear'); vq = F(xq);`
`bitshift(A,k,N)`	Warns	`bitshift(A, k, assumedtype)`	Replace all instances of `bitshift(A,k,N)` with `bitshift(A,k,assumedtype)`.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`bitcmp(A,N)`	Warns	`bitcmp(A, assumedtype)`	Replace all instances of `bitcmp(A,N)` with `bitcmp(A,assumedtype)`.
`repmat(A,r1,r2)`, where `r1` and `r2` are row vectors	Warns	`repmat(A,[r1 r2])`	Replace all instances of `repmat(A,r1,r2)` with `repmat(A,[r1 r2])`.
`repmat(A,empt)`, where `empt` is an empty array	Warns	`repmat(A,1)`	Replace all instances of `repmat(A,empt)` with `repmat(A,1)`.
`repmat(A,empt1,empt2)`, where `empt1` and `empt2` are empty arrays	Warns	`repmat(A,1)`	Replace all instances of `repmat(A,empt1,empt2)` with `repmat(A,1)`.
`repmat(A,n,empt)`, where `empt` is an empty array	Warns	`repmat(A,[n 1])`	Replace all instances of `repmat(A,n,empt)` with `repmat(A,[n 1])`.
`repmat(A,empt,n)`, where `empt` is an empty array	Errors	`repmat(A,[n 1])`	Replace all instances of `repmat(A,empt,n)` with `repmat(A,[n 1])`.
`repmat(A,col)`, where `col` is a column vector	Errors	`repmat(A,col.')`	Replace all instances of `repmat(A,col)` with `repmat(A,col.')`.
`repmat(A,B)`, where `B` is a matrix	Errors	`bvec = B(1:length(B)); repmat(A,bvec)`	Replace all instances of `repmat(A,B)` with the following code: `bvec = B(1:length(B)); repmat(A,bvec)`.
`repmat(A,B,C)`, where `B` and `C` are matrices	Errors	`BC = [B C]; bcvec = BC(1:length(BC)); repmat(A,bcvec)`	Replace all instances of `repmat(A,B,C)` with the following code: `BC = [B C]; bcvec = BC(1:length(BC)); repmat(A,bcvec)`.
Noninteger-valued size inputs for `zeros`, `ones`, `eye`, `Inf`, `NaN`, `true`, `false`, `rand`, `randi`, `randn`, and `cell`	Errors	Integer valued size inputs	Replace all instances of noninteger-valued size inputs with integer-valued size inputs for `zeros`, `ones`, `eye`, `Inf`, `NaN`, `true`, `false`, `rand`, `randi`, `randn`, and `cell`. You can use `floor` for this conversion.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`mimofr`	Errors	Not Applicable	Remove all instances of `mimofr` from your existing code.

Graphics

Mac support for copying figures in vector formats to other applications

MATLAB for Mac now supports copying figures to other applications in a high-resolution PDF format. If the figure contains uicontrols, then MATLAB uses a TIFF format instead.

savefig for saving figures to FIG-files

`savefig` saves one or more figures to a FIG-file.

OpenGL workarounds

If MATLAB crashes because of problems with your current version of OpenGL, then use one of the following workarounds:

- Start MATLAB with the `-softwareopengl` startup flag to use the software version of OpenGL.
- Execute the `opengl neverselect` command in your `startup.m` file to prevent MATLAB from selecting OpenGL as the renderer.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Numeric specifiers to set legend location	Still runs	Supported string specifiers	Remove all instances of using numeric specifiers to set the legend location
`get(0,'CommandWindowSize')`	Still Runs	`matlab.desktop.commandwindow.size`	Replace all instances of `get(0,'CommandWindowSize')` with `matlab.desktop.commandwindow.size`. The new command is compatible with MATLAB R2013a and later releases.
`printdlg('-setup',fig)`	Warns	Use the operating system printer management utilities to set up a new printer	Remove all instances of using `printdlg` with the `-setup` option

GUI Building

Custom icons for MATLAB apps you create

You can now package a custom icon in an app that you create. When selecting an icon, click **Use your own icon...** .

A second dialog box opens. Click **Select icon** to browse for a custom icon. The dialog box also displays three icon sizes. MATLAB automatically scales your icon for use in the Install dialog, App gallery, and quick access toolbar. For more information, see Package Apps.

Performance

repmat with numeric, char, and logical types

The performance of the `repmat` function improves when the input array contains numeric, char, or logical values.

Linear algebra functions on computers with new AMD processors

The performance of the linear algebra functions improves on computers with AMD® processors supporting the Intel AVX instruction set.

Data Import and Export

fprintf function prints Unicode characters to the screen

The `fprintf` function now displays Unicode characters on the screen. Previously, text was displayed with the user default character encoding, which is dependent on the user locale.

Compatibility Considerations

When displaying non-ASCII text on the screen, the `fprintf` function might return a different output value for the number of displayed bytes, compared to previous versions of MATLAB. This output value is now equal to the number of characters displayed on the screen.

There are no changes to the behavior of `fprintf` when writing text to a file, or when printing ASCII text to the screen.

Changes to default encoding for sendmail function

When calling the `sendmail` function on systems using a Japanese locale, the default character encoding is now UTF-8. Previously, the default encoding was SJIS.

Compatibility Considerations

This change affects only systems that use a Japanese locale. To use the SJIS character encoding, call:

```
setpref('Internet','E_mail_Charset','SJIS');
```

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`hdftool`	Warns		Read data from HDF files using `hdfread` or the low-level functions in the `matlab.io.hdf4.sd`, `matlab.io.hdfeos.gd`, and `matlab.io.hdfeos.sw` packages.

R2013a

Desktop

Option to add separators between controls on the quick access toolbar

You can now insert and move separators bars on the quick access toolbar. This helps to organize icons into groups. For information on how to customize and organize icons on the quick access toolbar, see Access Frequently Used Features.

Additional icon choices, auto-scaled thumbnails, and text-formatting options for customizing descriptions of MATLAB apps

When you package an app by clicking the **Package App** tab on the **Apps** tab, MATLAB opens the Package app dialog box. In the **Describe your app** section of the dialog box, you can:

- Click the app icon  , and then select a new icon from the display to represent your app in the Apps gallery.

- Use formatting options when writing the app description.

  Options include: bold, italic, monospace, and hyperlinked text; bulleted and numbered lists; images.

In addition, MATLAB automatically scales the screenshot you provide when packaging. This screenshot appears in the tooltip MATLAB displays when hovering over an app icon in the App gallery. The screenshot is scaled to fit a 300 x 300 area and maintains its original aspect ratio.

For more information, see Package Apps.

Left-aligned table of contents for navigating in the Help browser and online Documentation Center

The documentation now includes a table of contents that can remain open and that does not overlap the content. View the table of contents by clicking the Contents icon on the left edge of the window.

Search term highlighting and content expansion in the Help browser

In the Help browser, search terms now appear highlighted when you view pages linked from the search results. If the search term occurs within a collapsed section on the page, the browser automatically expands the content.

For example, the reference page for the `publish` function includes the term *bmp* in collapsed content. When you search the documentation for *bmp*, and then select the `publish` reference page, the Help browser expands the relevant section on the page and highlights all occurrences of the term.

To clear highlights, press the **Esc** key.

Context menu items in Help and Web browsers for zooming, page navigation, and saving

Both the Help browser and the MATLAB Web browser now provide easy access to commonly used features in their context menus. Right-click within the browser to see the available options, which include zooming to change the font size, searching within a page, navigating back or forward, or saving the HTML to a file.

Note These context menu items are not available on Macintosh systems.

Removal of Handle Graphics support under -nojvm startup option

When you start MATLAB with the `-nojvm` startup option, Handle Graphics® functionality is not supported. (The `-nojvm` option is available for UNIX platforms; see background information on the `matlab (UNIX)` reference page.)

Some aspects of MATLAB and related products use Handle Graphics in a way that might not be obvious. This includes anything that is based on or works using figures in MATLAB. Here is a summary of the affected features:

- Creating figures and performing plotting tasks, such as using the `plot`, `axes`, `getframe`, and `gcf` functions.
- Printing figures and using related functions such as `print`, `hgexport`, and `saveas`.
- Creating GUIs in MATLAB using GUI-building functions such as `warndlg`.
- Using Simulink scopes and printing Simulink models.

If you use the `-nojvm` startup option and use Handle Graphics functionality, MATLAB produces an error.

Compatibility Considerations

To avoid the error, start MATLAB without the `-nojvm` startup option:

- If you have been using the `-nojvm` startup option to work in a command line environment or because you do not use the MATLAB desktop, use the `-nodesktop` startup option instead.

- If you have been using the `-nojvm` startup option because of memory or performance benefits, look for other ways to gain those improvements when you start MATLAB without the `-nojvm` option. See topics in Performance and Memory.
- If you want to continue to use the `-nojvm` startup option, remove the code that is now producing the warnings.

-noFigureWindows startup option suppresses figures on Linux and Mac platforms

When you start MATLAB with the `-noFigureWindows` startup option, MATLAB disables the display of figure windows.

No default keyboard shortcut for overwrite mode

There is no longer a default keyboard shortcut for Overwrite mode, and **OVR** no longer appears at the bottom right corner of the MATLAB status bar. Previously, the **Insert** key by default toggled between Insert and Overwrite modes in the Command Window and in the MATLAB Editor.

Compatibility Considerations

To enable Overwrite mode, assign a keyboard shortcut to this action. On the **Home** tab, in the **Environment** section, click **Preferences > Keyboard > Shortcuts**. Search or scroll to find the action name, **Toggle Insert/Overwrite Mode**, and assign a keyboard shortcut. Use this shortcut to toggle between Insert and Overwrite modes.

Support requests using prerelease versions

If you have Internet access, you can submit a technical support request to MathWorks from MATLAB by selecting **Help > Request Support**. Previously, this feature was only available in the final version of each release. Beginning with R2013a, this feature is also available in the prerelease version.

Language and Programming

matlab.unittest package, an xUnit-style testing framework for the MATLAB language that allows writing and running unit tests, and analyzing test results

New MATLAB Unit Testing Framework (9 min, 24 sec)

For information about the `matlab.unittest` package, see Unit Testing Framework.

strsplit and strjoin functions for splitting and joining strings

The `strsplit` function splits a string into a cell array of strings. The `strjoin` function joins strings in a cell array into a single string.

Additional validateattributes options for checking array size and shape

The `validateattributes` function now checks the following attributes of input arrays.

Attribute	Description
`'3d'`	Array with three or fewer dimensions
`'ndims', N`	N-dimensional array
`'square'`	Square matrix
`'diag'`	Diagonal matrix

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

Help text for enumerations and events

The `help` and `doc` commands now display help text that you define for enumerations and events in custom classes. For more information, see Create Help for Classes.

Removal of support for .jar documentation files

The Help system no longer extracts documentation from `.jar` files. All custom documentation must be in uncompressed HTML files.

For more information, see Display Custom Documentation.

Changes to Microsoft .NET Framework support

As of R2013a, the MATLAB .NET interface requires the Microsoft .NET Framework Version 4.0 and above. The interface continues to support assemblies built on Framework 2.0 and above.

Compatibility Considerations

You must have the .NET Framework Version 4.0 or above installed on your system. If you enter any `NET.` or `System.` commands on a machine without Framework 4.0, MATLAB displays a warning. To determine if your system has a supported framework, use the `NET.IsNETSupported` function.

Changes to compiler support for building MEX-files

MATLAB supports these new compilers for building MEX-files.

Microsoft Windows platforms:

- Visual C++ 2012
- Intel C++ XE 2013
- Intel Visual Fortran XE 2013

Mac OS X platforms:

- Apple Xcode 4.2 and higher with Clang

MATLAB no longer supports the following compiler on Mac OS X platforms:

- Xcode with gcc

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

Changes to subclasses of built-in classes

Subclasses of built-in classes no longer inherit the following methods:

- `ones`
- `zeros`
- `true`
- `false`
- `cast`

As a result of this change, you cannot call these methods with an object of a built-in subclass. For example, given a class, `mySubclassOfDouble`, that derives from `double`, the following attempt to call `cast` results in an error:

```
a = mySubclassOfDouble(1);   % constructor
b = a.cast('int8');          % ok in R2012b but errors in R2013a
```

However, you can call the corresponding built-in functions with objects of the subclass. Use the function syntax instead of the method dot notation syntax:

```
b = cast(a,'int8');
```

Compatibility Considerations

Change code that attempts to call these methods to call the built-in function instead. Use function syntax instead of dot notation, as described in the previous section.

Conversion of error and warning message identifiers

For R2013a, error and warning message identifiers have changed in MATLAB.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `MATLAB:addtodate:Nargin` identifier has changed to `MATLAB:addtodatemx:Nargin`. If your code checks for `MATLAB:addtodate:Nargin`, you must update it to check for `MATLAB:addtodatemx:Nargin` instead. For a mapping of the new identifiers to the original identifiers, see Technical Support solution `1-ERAFNC`.

No strict-match requirements for month formats when converting date strings

When reading date strings, the date functions `datenum`, `datevec`, and `datestr` now treat months expressed as full names, three-letter abbreviations, or numbers in date string inputs interchangeably. In previous releases, the commands fail if a date string input does not match the format input. Using the `datestr` function to write date strings is not affected.

For example, consider the cell array of date strings, where each date string indicates the month in a different format:

```
mydates = {'31-October-2012','31-Oct-2012','31-10-2012'};
datevec(mydates,'dd-mm-yyyy')
```

```
ans =

        2012          10          31           0           0           0
        2012          10          31           0           0           0
        2012          10          31           0           0           0
```

Replacing the date string format input, `'dd-mm-yyyy'`, with `'dd-mmm-yyyy'` or `'dd-mmmm-yyyy'` returns the same output. Previously, the command failed, because the first two date strings include a full month name and a month abbreviation, which are not strict matches to the `mm` month format specified.

Compatibility Considerations

The date functions `datenum`, `datevec`, and `datestr` do not error when month formats are inconsistent across multiple date strings in a cell array of strings. When reading a cell array of date strings, you no longer can rely on using a format that includes a field for month name or abbreviation to catch date strings that specify a month number. You should ensure that all date strings in a cell

array have the same format to avoid the possibility of confusing day and month numbers. For example,

```
mydates = {'11-Oct-2012','11-10-2012'};
datevec(mydates,'dd-mmm-yyyy')

ans =

        2012          10          11           0           0           0
        2012          10          11           0           0           0
```

In R2013a, both dates are silently interpreted as October 11, 2012 although you might have intended `'11-10-2012'` to be in month-day-year format, representing November 10, 2012.

Date functions error on out-of-range quarter values

Valid values for quarter formats are Q1 through Q4 when converting date strings using the `datenum` or `datevec` functions. Values outside this range throw an error. Previously, the date functions accepted values outside the range, such as Q5, but returned incorrect results.

Compatibility Considerations

To prevent errors when parsing quarter values, remove values outside the valid range (Q1 through Q4) from your code.

String representations of large integers using exponential notation

The `num2str` function converts any large floating-point number that loses precision due to hardware limitations to a string representation that uses exponential notation. Such floating-point numbers are those greater than `flintmax`. Previously, `num2str` returned a string representation in decimal notation.

This change in behavior affects only the syntax `str = num2str(x)`. The following example compares the output of `num2str` in R2013a to R2012b.

R2013a	R2012b

`num2str(30e+25)`
 	
`ans =`
`3e+26`
	`num2str(30e+25)`
	
	`ans =`
	`30000000000000000000000000`

Compatibility Considerations

To obtain a string representation of large integers that uses decimal notation, call `int2str(x)` instead of `num2str(x)`.

Do not use classpath.txt file to modify Java static path

As of R2012b, the file `classpath.txt` is no longer used to add custom paths to the static Java class path. A new MathWorks product installation rewrites the static class path (found in the `classpath.txt` file).

Compatibility Considerations

To modify the Java class path, create `javaclasspath.txt` or `javalibrarypath.txt` files. For more information, see The Static Path and Locating Native Method Libraries.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`createCopy` method of `inputParser` class	Warns	`copy`	Replace all instances of `createCopy` with `copy`.
`hostid`	Warns		See Solution 1–171PI on the MathWorks website for instructions on obtaining your hostid.

Mathematics

scatteredInterpolant and griddedInterpolant support for extrapolation

- `scatteredInterpolant` is a new class for interpolating scattered data, and it returns extrapolated values by default when you evaluate at query points outside the convex hull.
- `griddedInterpolant` now returns extrapolated values by default when you evaluate at query points outside the domain of your sample grid.

Compatibility Considerations

`scatteredInterpolant` supports extrapolation by default. This behavior is different from that of `TriScatteredInterp`, which returns `NaN` when you evaluate at query points outside the convex hull using the `'linear'` or `'natural'` methods. To preserve the `TriScatteredInterp` behavior, set the `ExtrapolationMethod` property to `'none'`. See the `scatteredInterpolant` reference page for more information.

`griddedInterpolant` previously returned `NaN` values when you queried points outside the domain of the sample points using the `'linear'`, `'cubic'` or `'nearest'` interpolation methods. Now all interpolation methods support extrapolation by default. To preserve the old behavior, set the `ExtrapolationMethod` property to `'none'` when the interpolation method is `'linear'`, `'cubic'` or `'nearest'`. See the `griddedInterpolant` reference page for more information.

Syntax for ones, zeros, and other functions for creating arrays that match attributes of an existing variable

The functions `ones`, `zeros`, `eye`, `Inf`, `NaN`, `true`, `false`, and `cast` now can return an output that matches the data type, sparsity, and complexity (real or complex) of a variable `p`.

For example:

```
p = uint8([1 2]);
X = ones(2,3,'like',p);
class(X)

ans =

uint8
```

Integer type support for prod, cumsum, cumprod, median, mode, and number theory functions

The following functions now support inputs of any integer data type: `median`, `mode`, `prod`, `cumprod`, `cumsum`, `isprime`, `primes`, `factor`, `gcd`, `lcm`, `perms`, `factorial`, `nextpow2`, and `nchoosek`.

flintmax function for largest consecutive integer in floating-point format

Largest consecutive integer in floating-point format. `flintmax` returns the largest consecutive integer in IEEE® double precision, which is `2^53`. Above this value, double-precision format does not

have integer precision, and not all integers can be represented exactly. `flintmax('single')` returns the largest consecutive integer in IEEE single precision, which is `single(2^24)`.

Scale option for airy function

`scale` is an optional flag you can pass to `airy` to scale the resulting Airy function. See the `airy` reference page for more information.

scatteredInterpolant class that replaces TriScatteredInterp

`TriScatteredInterp` will be removed in a future release. Use the new `scatteredInterpolant` class instead. The `scatteredInterpolant` class performs interpolation on 2-D and 3-D scattered data with support for extrapolation outside the convex hull of the sample points. `scatteredInterpolant` also supports queries in grid vector format to conserve memory.

Compatibility Considerations

- `scatteredInterpolant` supports extrapolation by default. This behavior is different from that of `TriScatteredInterp`, which returns `NaN` when you evaluate at query points outside the convex hull using the `'linear'` or `'natural'` methods. To preserve the `TriScatteredInterp` behavior, set the `ExtrapolationMethod` property to `'none'`. See the `scatteredInterpolant` reference page for more information.
- `scatteredInterpolant` does not accept input of type `DelaunayTri`. Use `nearestNeighbor` or `pointLocation` to interpolate using a specific Delaunay triangulation. See Interpolation Using a Specific Delaunay Triangulation for more information.
- The `scatteredInterpolant` `Points` property is the array of sample point coordinates. The corresponding `TriScatteredInterp` property is called `X`. Use the `Points` property to refer to the sample points when you update your code.

triangulation class to replace TriRep

`TriRep` will be removed in a future release. Use the new `triangulation` class instead. The `triangulation` class represents 2-D and 3-D triangulations using a similar form and syntax as `TriRep`.

Compatibility Considerations

The property names and some method names have changed. The tables below map the current `TriRep` properties and methods to the `triangulation` replacements. Each replacement property has the same shape and holds the same values as the corresponding `TriRep` property. Similarly, each replacement method has the same syntax and returns the same result as the corresponding `TriRep` method.

Replacements for TriRep Properties

TriRep Property	Replacement triangulation Property	Name Change
`X`	`Points`	✓
`Triangulation`	`ConnectivityList`	✓

Replacements for TriRep Methods

TriRep Method	Replacement triangulation Method	Name Change
`baryToCart`	`barycentricToCartesian`	✓
`cartToBary`	`cartesianToBarycentric`	✓
`circumcenters`	`circumcenter`	✓
`edgeAttachments`	`edgeAttachments`	
`edges`	`edges`	
`faceNormals`	`faceNormal`	✓
`featureEdges`	`featureEdges`	
`freeBoundary`	`freeBoundary`	
`incenters`	`incenter`	✓
`isEdge`	`isConnected`	✓
`neighbors`	`neighbors`	
`size`	`size`	
`vertexAttachments`	`vertexAttachments`	

delaunayTriangulation class to replace DelaunayTri

`DelaunayTri` will be removed in a future release. Use the new `delaunayTriangulation` class instead. `delaunayTriangulation` represents 2-D and 3-D Delaunay triangulations using a similar form and syntax as `DelaunayTri`.

Compatibility Considerations

Some of the property and method names have changed. The tables below map the current `DelaunayTri` properties and methods to the `delaunayTriangulation` replacements. Each replacement property has the same shape and holds the same values as the corresponding `DelaunayTri` property. Similarly, each replacement method has the same syntax and returns the same result as the corresponding `DelaunayTri` method.

Replacements for DelaunayTri Properties

DelaunayTri Property	Replacement delaunayTriangulation Property	Name Change
`Constraints`	`Constraints`	
`X`	`Points`	✓
`Triangulation`	`ConnectivityList`	✓

Replacements for DelaunayTri Methods

DelaunayTri Method	Replacement delaunayTriangulation Method	Name Change
`convexHull`	`convexHull`	
`inOutStatus`	`isInterior`	✓
`nearestNeighbor`	`nearestNeighbor`	
`pointLocation`	`pointLocation`	
`voronoiDiagram`	`voronoiDiagram`	

Set functions behavior change

The behavior of `unique`, `union`, `intersect`, `setdiff`, `setxor`, and `ismember` has changed.

- If there are repeated elements in the input arrays, the functions `unique`, `union`, `intersect`, `setdiff`, and `setxor` return the index to the first occurrence of the repeated elements (or rows).
- The optional output array, `locb`, returned by `ismember`, contains the lowest absolute indices in `B` for elements (or rows) that are also in `A`.
- All index vectors returned by `unique`, `union`, `intersect`, `setdiff`, or `setxor` are column vectors.
- The shape of the output, `C`, from `intersect`, `setxor`, and `union`, when the `'rows'` and `'legacy'` flags are not specified, is as follows. `C` is column vector unless both `A` and `B` are row vectors, in which case `C` is a row vector.
- The shape of the output, `C`, from `setdiff`, when the `'rows'` and `'legacy'` flags are not specified, is as follows. `C` is a row vector if `A` is a row vector. Otherwise, `C` is a column vector.
- `unique`, `union`, `intersect`, `setdiff`, `setxor`, and `ismember` support objects.
- The input arrays passed to `union`, `intersect`, `setdiff`, `setxor`, and `ismember` must be of the same class with the following exceptions:
  - Logical, char, and all numeric classes can combine with double arrays.
  - Cell arrays of strings can combine with char arrays.
- `ismember` treats trailing white space in cell arrays of strings as distinct characters. For example, `'word'` is different from `'word '`. If the `'legacy'` flag is specified, `ismember` ignores trailing white space and treats `'word'` the same as `'word '`.

Compatibility Considerations

If the changes adversely affect your code, you can specify `'legacy'` to preserve the behavior from R2012b and prior releases. For example:

```
[C,IA,IC] = unique([9 9 1])

C =

     1     9


IA =

     3
     1


IC =

     2
     2
     1

[C2,IA2,IC2] = unique([9 9 1],'legacy')

C2 =

     1     9


IA2 =

     3     2


IC2 =

     2     2     1
```

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`quad`	Still runs	`integral`	Replace all instances of `quad` with `integral`. For example, replace `q = quad('x.*cos(x)',0,1)` with `q = integral(@(x)x.*cos(x),0,1)`. If `quad` uses an absolute error tolerance, `tol`, replace all instances of the tolerance argument with the `'AbsTol'` or `'RelTol'` name-value pair arguments. For example, replace `q = quad(fun,a,b,tol)` with `q = integral(fun, a, b, 'AbsTol', tol, 'RelTol', tol)`.
`quadl`	Still runs	`integral`	Replace all instances of `quadl` with `integral`. For example, replace `q = quadl('x.*cos(x)',0,1)` with `q = integral(@(x)x.*cos(x),0,1)`. If `quadl` uses an absolute error tolerance, `tol`, replace all instances of the tolerance argument with the `'AbsTol'` or `'RelTol'` name-value pair arguments. For example, replace `q = quadl(fun,a,b,tol)` with `q = integral(fun, a, b, 'AbsTol', tol, 'RelTol', tol)`.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`quadv`	Still runs	`integral` with the `'ArrayValued',true` name-value pair argument	Replace all instances of `quadv` with `integral` using the `'ArrayValued',true` name-value pair argument.
`dblquad`	Still runs	`integral2`	Replace all instances of `dblquad` with `integral2`. If there are discontinuities in the interior of the region of integration, replace `dblquad` with `integral2` using the `'method','iterated'` name-value pair argument.
`triplequad`	Still runs	`integral3`	Replace all instances of `triplequad` with `integral3`. If there are discontinuities in the interior of the region of integration, replace `triplequad` with `integral3` using the `'method','iterated'` name-value pair argument.
`bitmax`	Still runs	`flintmax`	Replace all instances of `bitmax` with `flintmax`.
`TriRep`	Still runs	`triangulation`	Replace all instances of `TriRep` with `triangulation`. Most of the `triangulation` properties and methods are the same as those in `TriRep`, but there are a few exceptions. See “triangulation class to replace TriRep” on page 16-11 for details.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`DelaunayTri`	Still runs	`delaunayTriangulation`	Replace all instances of `DelaunayTri` with `delaunayTriangulation`. Most of the `delaunayTriangulation` properties and methods are the same as those in `DelaunayTri`, but there are a few exceptions. See “delaunayTriangulation class to replace DelaunayTri” on page 16-12 for details.
`TriScatteredInterp`	Still runs	`scatteredInterpolant`	Replace all instances of `TriScatteredInterp` with `scatteredInterpolant`. See “scatteredInterpolant class that replaces TriScatteredInterp” on page 16-11 for compatibility details.
`inline`	Still runs	Anonymous function	Replace all instances of `inline` with an anonymous function. See Anonymous Functions for more information. You can use `symvar` and `func2str` to convert your existing code.
`rand` or `randn` with the `'seed'`, `'state'` or `'twister'` inputs.	Still runs	`rng`	For more information about updating your code and assessing the impact of this change, see Updating Your Random Number Generator Syntax.
`permute(A,order)` where `order` contains noninteger or complex values.	Warns	Real, positive integer values for `order`	Replace all instances of noninteger or complex valued inputs with real, positive integer values. You can use `round` or `real` for this conversion.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
Noninteger valued size inputs for `zeros`, `ones`, `eye`, `Inf`, `NaN`, `true`, `false`, `rand`, `randi`, `randn`, and `cell`	Warns	Integer valued size inputs	Replace all instances of noninteger valued size inputs with integer valued size inputs for `zeros`, `ones`, `eye`, `Inf`, `NaN`, `true`, `false`, `rand`, `randi`, `randn`, and `cell`. You can use `floor` for this conversion.
`cholinc(X,'inf')`	Errors	None	Remove all instances of `cholinc(X,'inf')` from your code.
All other syntaxes of `cholinc` except `cholinc(X,'inf')`	Errors	`ichol`	Replace these instances of `cholinc` with `ichol`.
`luinc`	Errors	`ilu`	Replace all instances of `luinc` with `ilu`.
`sparse(A)` `sparse(i,j,s,...)` where `A` and `s` are of type `char`.	Errors	`sparse(double(A))` `sparse(i,j,double(s),...)`	Convert all `char` inputs to `double` before calling `sparse`.
`RandnAlg` property of `RandStream` class	Errors	`NormalTransform` property of `RandStream` class	Replace all existing instances of `RandnAlg` with `NormalTransform`.
`setDefaultStream` method of `RandStream` class	Errors	`setGlobalStream` method of `RandStream` class	Replace all existing instances of `RandStream.setDefaultStream` with `RandStream.setGlobalStream`.
`getDefaultStream` method of `RandStream` class	Errors	`getGlobalStream` method of `RandStream` class	Replace all existing instances of `RandStream.getDefaultStream` with `RandStream.getGlobalStream`.

Graphics

gobjects function for preallocating graphics handle array

The `gobjects` function creates an array of graphics handles. Use this function instead of the `ones` or `zeros` function when preallocating an array to store graphics handles.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Figure `DoubleBuffer` property	No effect	No replacement needed	On older computer systems, was used to produced flash-free graph animations.
Use of `ginput` when MATLAB is started with the `-noFigureWindow` or `-nodisplay` flag	Errors	Not applicable	Do not use `ginput` when you start MATLAB with the `-noFigureWindows` or `-nodisplay` flags.
Use of graphics format extensions `ai` or `ill` with `saveas`	Warns	No replacement	Remove all instances of `saveas` with the `ai` or `ill` extensions.
Use of `all` option with `hgsave` or `hgload`	Warns	No replacement	Remove all instances of passing `all` to `hgsave` or `hgload`.
Use of `getframe` with second input argument that specifies an area not contained in figure window	Warns	No replacement	If using a second input argument with `getframe`, then specify an area fully contained in figure window.
Use of `-dsetup` option with `print` to display the Windows Print Setup dialog	Warns	No replacement	Remove all instances of `-dsetup`. For the current figure, click **File > Print** in the Figure window to display the Print Setup dialog.

Data Import and Export

Reading and writing indexed and grayscale AVI files with VideoReader and VideoWriter objects

The `VideoReader.read` method now returns an array of grayscale data when reading grayscale AVI files. The `read` method also reads AVI files with indexed video and a colormap using the new `'native'` input argument.

The `VideoWriter.writeVideo` method now writes grayscale AVI files, and AVI files with indexed video and a colormap.

Compatibility Considerations

A `VideoReader` object created from an AVI file with indexed video and a grayscale colormap now has a `VideoFormat` property with the value `'Grayscale'`. By default, using the `VideoReader.read` method to read this object returns an `m`-by-`n`-by-1-by-`F` array, where `m` is the image frame height, `n` is the image frame width, and `F` is the number of frames read. In R2012b and earlier releases, the value of `VideoFormat` was `'RGB24'` and `read` by default returned an `m`-by-`n`-by-3-by-`F` array.

Writing MPEG-4 H.264 files on Mac with VideoWriter object

`VideoWriter` now writes MPEG-4 H.264 files with the extension `.mp4` or `.m4v` on systems with Mac OS X 10.7 or later.

Tiff object improvements for reading and writing RGB-class TIFF images

The `Tiff.readRGBAImage` method returns RGBA data from an entire RGB-compatible image. The `Tiff.readRGBAStrip` and `Tiff.readRGBATile` methods return RGBA data from a single strip or a single tile in an RGB-compatible image, respectively.

The `JPEGColorMode` property for `Tiff` objects controls YCbCr/RGB conversion when writing YCbCr images. For more information, see the `Tiff` reference page.

Importing non-ASCII encoded files with textscan function

The `textscan` function now reads text files with non-ASCII encodings. For a complete list of supported encoding schemes and the syntax for specifying the encoding, see the `fopen` reference page.

Multichannel JP2 support in imread function

By default, when reading JP2 files, the `imread` function now returns all image channels in the order in which they exist in the codestream. In R2012b and earlier releases, `imread` read up to 3 channels only.

Compatibility Considerations

The default syntax of `imread` no longer reorders the channels of a JP2 image where the channels are out of order. To reorder the channels, call `imread` with the `'V79Compatible'` name-value pair argument.

Previous behavior change of xlsread function output

In R2011b and earlier, the `xlsread` function returned columns of empty strings, `''`, in the text data output, where there were leading columns of numeric data preceding text data. As of R2012a, on systems with Excel for Windows, these columns of empty strings are trimmed from the text data output.

Compatibility Considerations

This change in behavior affects file reading on systems with Excel for Windows, when calling `xlsread` without the `'basic'` argument. For example, given a file, `myfile.xls` , where Sheet1 contains the following data:

```
1    1    1    A    A
2    2    2    B    B
3    3    3    C    C
4    4    4    D    D
5    5    5    E    E
```

Calling `[num,txt] = xlsread('myfile.xls','Sheet1')` on systems with Excel for Windows returns:

```
num =
     1     1     1
     2     2     2
     3     3     3
     4     4     4
     5     5     5


txt =
    'A'    'A'
    'B'    'B'
    'C'    'C'
    'D'    'D'
    'E'    'E'
```

In R2011b and earlier, the command returned:

```
num =

     1     1     1
     2     2     2
     3     3     3
     4     4     4
     5     5     5


txt =
```

```
    ''    ''    ''    'A'    'A'
    ''    ''    ''    'B'    'B'
    ''    ''    ''    'C'    'C'
    ''    ''    ''    'D'    'D'
    ''    ''    ''    'E'    'E'
```

When reading XLS files, you can use the `'basic'` flag to reproduce the pre-R2012a output:

```
[num,txt] = xlsread('filename.xls',Sheet,'','basic');
```

Authentication, user name, and password inputs for urlread and urlwrite functions

The `urlread` and `urlwrite` functions accept the following optional name-value pair arguments.

Attribute	Value
`Authentication`	HTTP authentication mechanism. Currently, only basic authentication is supported.
`Username`	User identifier.
`Password`	User authentication string.

Additional audio and video file reading capabilities using Import Wizard and importdata function

The Import Wizard and the `importdata` function now import all audio file formats supported by the `audioread` function, including WAV, FLAC, and OGG files on all platforms, and MP3 and MPEG-4 AAC files on Windows 7 (or later), Macintosh, and Linux platforms.

The Import Wizard and the `importdata` function now import all video file formats supported by the `VideoReader` class.

sound function nonblocking

The `sound` function now returns immediately. In R2012b and earlier releases, `sound` blocked until all audio played back.

Compatibility Considerations

To play audio with blocking, use `playblocking(audioplayer(y,Fs))` instead of `sound(y,Fs)`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`aufinfo`	Still runs	`audioinfo`	Replace all instances of `aufinfo` with `audioinfo`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`auread`	Still runs	`audioread`	Replace all instances of `auread` with `audioread`.
`auwrite`	Still runs		Write audio files using `audiowrite`.
`wavfinfo`	Still runs	`audioinfo`	Replace all instances of `wavfinfo` with `audioinfo`.
`wavread`	Still runs	`audioread`	Replace all instances of `wavread` with `audioread`.
`wavwrite`	Still runs	`audiowrite`	Replace all instances of `wavwrite` with `audiowrite`.
`cdfwrite`	Still runs		Write Common Data Format (CDF) files using the `cdflib` low-level functions.
`hdftool`	Still runs		Read data from HDF files using `hdfread` or the low-level functions in the `matlab.io.hdf4.sd`, `matlab.io.hdfeos.gd`, and `matlab.io.hdfeos.sw` packages.

Performance

fft function performance improvements on computers with new Intel and AMD processors

The `fft` function performance improves on computers with Intel and AMD processors supporting the AVX instruction set.

permute function performance improvements for 3-D and higher dimensional arrays

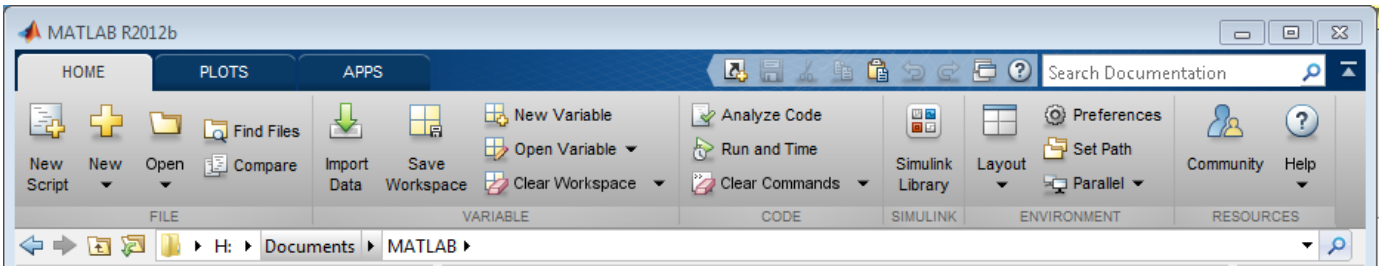
The `permute` function performance improves for 3-D and higher dimensional arrays.

R2012b

Desktop

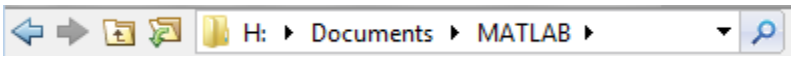
Toolstrip that replaces menus and toolbars in MATLAB Desktop

New MATLAB Desktop (5 min, 3 sec)

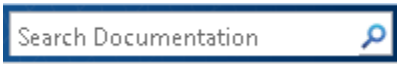


- The Toolstrip contains components that were previously available in menus, toolbars, and the **Start** button. Tabs such as **Home**, **Plots**, **Apps**, **Editor**, and **Variable**, group functionality to support common tasks.
- The **Apps** tab contains a gallery of apps from the MATLAB family of products. You also can install your own apps, which appear in the apps gallery.
- The quick access toolbar contains frequently used options such as cut, copy, paste. The quick access toolbar is customizable; you can add items from tabs or from command shortcuts you create. You can also change the position of the toolbar on the Desktop.

- The current folder toolbar enables you to control the current working directory. The location and the controls available are customizable.



- The **Search Documentation** box enables you to search the documentation for functions and other topics of interest.



Apps gallery that presents apps from the MATLAB product family

The MATLAB desktop contains a gallery of apps from the MATLAB family of products. You can also install your own apps, which appear in the apps gallery.

- To access the apps gallery, on the desktop Toolstrip, click the **Apps** tab, and then, on the far right of the **Apps** section, click the arrow ▾.
- To find and install apps written by others, click the **Apps** tab, and in the **File** section, click **Get More Apps**.

Single-file application packaging as a MATLAB App Installer file for inclusion in the apps gallery

Packaging and Installing MATLAB Apps (2 min, 58 sec)

You can package your GUI as a single installer file that contains all of the code and data needed to run your app, as well as a description of your app. Share your app with others by uploading the installer file to File Exchange or sending it as an email attachment. When your app installs, it appears alongside apps from the MathWorks family of products in the apps gallery. For more information, see Package Apps.

Redesigned Help with improved browsing, searching, and filtering

Redesigned Help (5 min, 43 sec)

Searching and navigation

The R2012b release includes a redesigned documentation system with streamlined product pages, detailed search suggestions, and enhanced filtering of search results.

Each product's documentation is now organized by categories of functionality, such as Graphics or Simulation, rather than by information type, such as Function Reference or User's Guide. Categories include links to related reference pages, examples, and conceptual topics, allowing you to browse from one to another.

Most features of the previous Help browser are available in the new browser, although some functionality has a different appearance or location. For example:

- The table of contents for each product is collapsed by default to maximize the space available for reading topics. Expand the table of contents from any page in the Help browser by clicking the Table of Contents button,  .
- Links to alphabetical lists of reference pages, such as functions or blocks, appear at the bottom of each product page.
- Tasks previously performed using menu items in the Help browser, such as setting preferences or viewing page locations, are now accessible via context (right-click) menus, keyboard shortcuts, or toolbar buttons.

  Note On Macintosh systems, there are no context menu items for **Evaluate Selection** or **Get Page Address**. You can copy selected code to the Command Window for evaluation by pressing **Shift+F7**. However, in this release, there is no way to identify the page address.

- Demos are now labeled *Examples*, and are accessible from the top of each product page. When you select video examples, they open in your system Web browser.

- All release notes for a product appear on the same page, allowing you to track changes across several releases from a single location.

The new Help browser is used exclusively for the documentation of MathWorks products. As a result:

- The `web` command opens all pages that are not part of the documentation in the MATLAB Web browser, even when you specify the `-helpbrowser` option. Because this is the default behavior of the `web` command, the `-helpbrowser` option has been removed from the documentation.
- When the `doc` command does not find a documentation page, but does find associated help text, it displays the text in the MATLAB Web browser rather than the Help browser.
- Custom documentation and documentation for downloadable or third-party products displays in a separate browser. To access this browser, open the Help browser and navigate to the documentation home page. Then, at the bottom of the page, click **Supplemental Software**.

Font size in Help browser and Web browser

The Help browser and Web browser now allow you to zoom in and out to adjust the font size by pressing **Ctrl** and **+** or **–**.

In-product access to online documentation

From within the product, you can access either the installed documentation or the documentation on the Web. By default, your Help preference is set to view the installed documentation. If you change the preference to view the Web documentation, then you can choose to view documentation for products that you do not have installed.

Searches using the doc command

In previous releases, if you passed a term to the `doc` command that did not correspond to the name of a MathWorks reference page or to a file with help text, the `doc` command issued an error. Now, the `doc` command searches the documentation for the term, and displays the search results in the Help browser.

Improved rendering in Help browser and Web browser

The new Help browser uses a different rendering engine than the previous Help browser. This engine provides improved legibility, particularly on Microsoft Windows 64-bit systems.

The MATLAB Web browser also uses this rendering engine on all platforms. In previous releases, the Web browser used this engine only on Microsoft Windows systems.

This rendering engine is not fully supported on Red Hat® 5 operating systems, so not all Help features are available on those systems. For example:

- Search suggestions are available from the search bar in the desktop, but not within the Help browser.
- Some components of the search results page are not available, such as filtering.
- The Help browser can display installed product documentation, but not the documentation on the Web.

Compatibility Considerations

- The Help system now requires that custom documentation files reside outside the `matlabroot` folder (but on the MATLAB search path). To view any installed custom documentation, open the Help browser and navigate to the documentation home page. Then, at the bottom of the page, click **Supplemental Software**.
- The `demo` command no longer supports categories of MATLAB or Simulink examples.
- For some products (primarily third-party toolboxes), the `demo` command requires different input values to specify the product type or name. The `demo` command now identifies each product using its `info.xml` file rather than its `demos.xml` file. For most MathWorks products, the product information is the same in both files.
- In a future release, MathWorks will remove support for `demos.m` files, an old way to include demos in the Help browser. For the recommended method, see Display Custom Examples.

Viewing of multiple documentation pages simultaneously with tabbed browsing

The Help browser now allows you to open multiple tabs so that you can view more than one documentation topic at a time. Open a new tab by clicking the New Tab button,  , or by right-clicking a link.

Suggested corrections for mistyped functions and variables in the Command Window

Command Window Suggestions for Mistyped Functions and Variables (2 min, 1 sec)

MATLAB can show suggestions in the Command Window for misspelled functions and variable names. If you enter an undefined function or variable name, MATLAB displays:

```
Did you mean:
```

followed by a suggested command at the command line. You can press **Enter** to execute that command, or **Esc** to delete the suggestion.

Full-screen view mode on Mac operating systems

MATLAB can operate in full-screen mode on Mac OS X 10.7 Lion.

Changes to -nojvm startup option on Mac

When you start MATLAB on a Mac with the `-nojvm` startup option, Handle Graphics functionality is no longer supported. MATLAB will produce an error. Previous to this release, calls to Handle Graphics functionality would work but generate a warning. (For more information about the `-nojvm` option, see the `matlab (UNIX)` reference page.)

Tabs in MATLAB Web browser

The MATLAB Web browser now allows you to open multiple tabs so that you can view multiple reports or Web sites simultaneously. Open a new tab by clicking the New Tab button,  .

Direct access to run configurations from the Run button

Now you can type commands directly using the **Run** options on the **Editor** tab.

In previous versions, you accessed run configurations by clicking the **Run** button down arrow and selecting **Edit Run Configurations for *`filename`***.

Compatibility Considerations

MATLAB R2012b does not automatically copy run configurations from versions prior to R2012b. To transfer run configurations from a release prior to the MATLAB R2012b installation:

1. Open `prefdir\run_configurations.m`. To identify your preference directory, type `prefdir` in the Command Window.

   Run configuration entries appear similar to this example:

   ```
   %% @name foo
   %  @associatedFile C:\Documents\MATLAB\work\foo.m
   %  @mostRecentlyActioned true
   %  @uniqueId 695644b0:1355f0bb372:-7fb0

   % Modify expression to add input arguments.
   % Example:
   %   a = [1 2 3; 4 5 6];
   %   foo(a);
   foo(1,2)
   ```

2. Copy the MATLAB command in the run configuration. In the example configuration, the command is `foo(1,2)`.
3. Open the MATLAB file associated with the run configuration that you want to keep. In the example configuration, the file is `C:\Documents\MATLAB\work\foo.m`.
4. Click  and then the field containing `type code to run`.
5. Paste the MATLAB command.

If the MATLAB expression spans multiple lines, create a script, and copy and paste the code into the script. You can then run the script directly using  .

Multiple vector creation from single selection in Variables editor

The Variables editor now allows you to create one or more new row or column vectors from a single selection within an existing variable. The names of the new variables are based on the name of the

existing variable. Previously, the Variables editor could only create a new array from a single selection, and new variables were called `unnamed`, `unnamed1, …, unnamedN`.

Language and Programming

Abstract attribute for declaring MATLAB classes as abstract

Declare MATLAB classes as abstract by setting the class `Abstract` attribute. See Defining Abstract Classes for more information.

Diagnostic message improvements when attempting to create an instance of an abstract class

Attempting to instantiate an abstract class returns a list of abstract members and their defining classes in the error diagnostic message. The `meta.abstractDetails` function finds abstract members defined or inherited by abstract classes.

Handle and dynamicprops do not support the empty static method

For R2012b, the `handle` and `dynamicprops` classes no longer support the `empty` static method. This change makes these classes consistent with all MATLAB abstract classes.

Compatibility Considerations

MATLAB software issues an error for calls to the `empty` static method from the `handle` and `dynamicprops` classes:

```
% Will return errors
handle.empty
dynamicprops.empty
```

See Creating Empty Arrays for information on using `empty`.

Switch uses eq to compare enumerations

`Switch` statements use the enumeration `eq` method to compare the `switch` expression with the `case` expression (see `switch`). In previous releases, `switch` used a special comparison that required the result of both expressions to belong to the same enumeration class. The result of the comparison is determined by the result of:

```
switch_expression == case_expression
```

The new behavior is consistent with that of other conditional statements, such as `if` statements.

Compatibility Considerations

Code that uses enumerations in `switch` statements can behave differently in release R2012b than in a previous release.

Cannot specify property attributes multiple times

Classes can define a property attribute only once in a properties block. Defining the same attribute more than once in a properties block causes an error when the class is instantiated.

Compatibility Considerations

Classes that define the same property attribute multiple times in a properties block will error when attempting to instantiate the class. Previously, MATLAB evaluated property attribute specifications in right to left order.

Discontinued compiler support for building MEX-files

MATLAB no longer supports the following compilers on Linux 32-bit platforms:

- GNU gcc Version 4.4.x
- GNU gfortran 4.3.x

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see Supported and Compatible Compilers.

Jagged array support for .NET

- To create a .NET jagged array, use the `NET.createArray`.
- To convert a .NET jagged array to a MATLAB array, see How MATLAB Handles Jagged Arrays.
- To convert a MATLAB array to a .NET jagged array, see Pass MATLAB Arrays as Jagged Arrays.

MATLAB does not support:

- Autoconversion of `char` or cell arrays to jagged array arguments. For more information, see Call .NET Method with System.String Jagged Array Arguments.
- Autoconversion of MATLAB arrays to multidimensional jagged array arguments. For more information, see Call .NET Method with Multidimensional Jagged Array Arguments.

Java exceptions accessible to MATLAB code

If you call a Java method from MATLAB, and that code throws an exception, use the `matlab.exception.JavaException` class to handle the exception in MATLAB.

Ability to add jar files to static Java class path

To use third-party Java libraries in MATLAB, you can control the Java class path and native library path by creating `javaclasspath.txt` and `javalibrarypath.txt` files. For more information, see The Static Path and Locating Native Method Libraries.

Preservation of string functions for backwards compatibility

The R2010a Release Notes originally stated that the `isstr`, `setstr`, `str2mat`, `strread`, `strvcat`, and `textread` functions would be removed in a future release. As of R2012b, there are no plans to remove these functions. However, use of these functions is not recommended.

Function	Recommended Modification
`isstr`	Replace all existing instances of `isstr` with `ischar`.
`setstr`	Replace all existing instances of `setstr` with `char`.
`str2mat`	Replace all existing instances of `str2mat` with `char`.

`strread`	Replace all existing instances of `strread` with `textscan`. For example, replace `[a,b,c] = strread(...)` with
	`C = textscan(...)`
	`[a,b,c] = deal(C{:})`
	Unlike `strread`, the `textscan` function converts numeric values to the specified data type, allowing preservation of integer types.

`strvcat`	Replace all existing instances of `strvcat` with `char`. Unlike `strvcat`, the `char` function does *not* ignore empty strings.
`textread`	Replace all existing instances of `textread` with `textscan`, similar to `strread`. Open and close files with `fopen` and `fclose`.

Conversion of Error and Warning Message Identifiers

For R2012b, error and warning message identifiers have changed in MATLAB.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `MATLAB:uitable:InvalidParent` identifier has changed to `MATLAB:uitable:ParentMustBeFigureOrUIContainer`. If your code checks for `MATLAB:uitable:InvalidParent`, you must update it to check for `MATLAB:uitable:ParentMustBeFigureOrUIContainer` instead. For a mapping of the new identifiers to the original identifiers, see Technical Support solution `1-ERAFNC`.

Mathematics

Performance improvements and multithreading for airy, psi, and Bessel functions

The following functions show improved performance: `besselh`, `besseli`, `besselj`, `besselk`, `bessely`, `airy`, and `psi`.

ddensd function that solves delay differential equations of neutral type with state-dependent delays

The `ddensd` function solves neutral delay differential equations that involve both the solution and its first derivative evaluated in the past:

$$y'(t) = f(t, y(t), y(dy_1),...,y(dy_j), y'(dyp_1),...,y'(dyp_k))$$

The delays, *dy* and *dyp*, can be time-dependent, state-dependent, or both.

Signed integer support for bit-wise operations

The `bitand`, `bitor`, `bitxor`, `bitcmp`, `bitshift`, `bitget`, and `bitset`, functions now support signed integers.

An additional input argument, `assumedtype`, is an optional string argument you can pass to any of these functions. Use `assumedtype` to indicate the assumed integer class for input values of type `double`. For example, `bitor(14,240,'uint8')` treats `14` and `240` as unsigned 8-bit integers even though they are passed as integers of type `double`.

`assumedtype` can be one of the following strings: `'int8'`, `'uint8'`, `'int16'`, `'uint16'`, `'int32'`, `'uint32'`, `'int64'`, or `'uint64'`. The default value of `assumedtype` is `'uint64'` for input values of type `double`. If the input values belong to an integer class, `assumedtype` defaults to the class of the input values.

atan2d function that calculates four-quadrant inverse tangent with result in degrees

`atan2d` calculates the four-quadrant inverse tangent and returns angles in degrees that lie in the closed interval [–180,180].

Complex number support for trigonometry degree functions

The `sind`, `cosd`, `tand`, `asind`, `acosd`, `atand`, `cscd`, `cotd`, `secd`, `acscd`, `asecd`, and `acotd` functions now support complex values. For example, `sind(10+i)` returns `0.1737 + 0.0172i`. Likewise, `asind(2)` now returns the complex angle `90.0000 -75.4561i`.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`bitshift(A,k,N)`	Still Runs	`bitshift(A,k,assumedtype)`	Replace all instances of `bitshift(A,k,N)` with `bitshift(A,k,assumedtype)`.
`bitcmp(A,N)`	Still Runs	`bitcmp(A,assumedtype)`	Replace all instances of `bitcmp(A,N)` with `bitcmp(A,assumedtype)`.
`bitshift(A,k)` where A is of type`double`.	Still runs	Not Applicable	`bitshift` now interprets `double` input as `uint64` instead of 53-bit signed integer by default. For example, `bitshift(1,54)` now returns `1.8014e+16` instead of `0`.
`mimofr`	Warns	Not Applicable	Remove all instances of `mimofr` from your existing code.
Second output argument for `besselh`, `besseli`, `besselj`, `besselk`, `bessely`, and `airy`. For example, `[J,ierr] = besselj(nu,Z)`.	Errors	Syntax that returns only the solution vector. For example, `J = besselj(nu,Z)`.	Replace all instances that return two output arguments with the syntax that returns only the solution vector.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed-orientation input vectors to `besselh`, `besseli`, `besselj`, `besselk`, and `bessely`. For example, passing a row vector followed by a column vector:
`J= besselj(rowNu, colZ)`
or passing a column vector followed by a row vector:
`J= besselj(colNu, rowZ)`
	Errors	First construct the inputs with `ndgrid` or `meshgrid`. Alternatively, you can pass a function handle and the mixed-orientation vectors to `bsxfun`.	Modify all instances that pass mixed-orientation vectors. You can modify your code in one of two ways:
			• Call `meshgrid` or `ndgrid` to construct the full grid first.
			`[nu,Z]= meshgrid(rowNu, ColZ);`
			`J= besselj(nu, Z);`
			• Pass a function handle and the mixed-orientation vectors to `bsxfun`. For example, if your existing code passes a row vector followed by a column vector, make the following change:
			`J= bsxfun(@besselj, rowNu, colZ);`or, if your code passes a column vector followed by a row vector, make the following change:
			`J= bsxfun(@besselj, colNu', rowZ.');`

`Y = psi(k0:k1,X)`	Errors	`Y = psi(k,X)` where `k` is a scalar specifying the `k`th derivative of ψ at the elements of `X`.	Replace all instances of `Y = psi(k0:k1,X)` with `Y = psi(k,X)`, where `k` is a scalar. To modify your code, loop through the values `k0:k1`. For example:
			`for k=k0:k1`
			`  Y(:,k) = psi(k,X);`
			`end`
			In the future, `size(Y)` will be `size(X)`. Modify any code that depends on `size(Y)`.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing empty and nonscalar input to `besselh`, `besseli`, `besselj`, `besselk`, `bessely`, and `airy`. For example,
`J = besselj([],(1:3))`
or
`J = besselj((1:3),[])`
	Errors	`J = besselj(nu,[])` or `J = besselj([],Z)` where `nu` and `Z` are scalars.	Modify all instances that pass combinations of empty arrays with nonscalar input. The inputs must be the same size or one must be a scalar.

Data Import and Export

Data import from delimited and fixed-width text files using Import Tool

The Import Tool now allows you to preview data in delimited and fixed-width text files and select ranges of data to import. You also can define rules for handling nonnumeric values, and import data as column vectors, a matrix, a cell array, or a dataset array, in a single step. This tool opens instead of the Import Wizard for text files.

For more information, watch this video.

Single-step import of numbers, text, and dates as column vectors from a spreadsheet with Import Tool

The Import Tool now allows you to import columns of numeric data, text, and dates from spreadsheets as multiple column vectors in one step. Previously, this functionality was available for numeric data only.

audioread and audioinfo functions for reading MP3, MPEG-4 AAC, WAVE, and other audio files

The `audioread` and `audioinfo` functions read and provide information about MP3, MPEG-4 AAC, WAVE, OGG, and FLAC files on all platforms.

`audioread` is a drop-in replacement for the most common forms of `wavread` and `auread`. In most cases where you use `wavread` or `auread`, you can rename instances of these functions to `audioread`.

audiowrite function for writing MPEG-4 AAC, WAVE, and other audio files

The `audiowrite` function writes data to WAVE, OGG, and FLAC files on all platforms. `audiowrite` writes data to MPEG-4 AAC files on Windows and Mac platforms.

`audiowrite` is a drop-in replacement for the most common forms of `wavwrite`. In most cases where you use `wavwrite`, you can rename instances of this function to `audiowrite`.

Reading and writing of BigTIFF image files larger than 4 GB

MATLAB now supports BigTIFF image files larger than 4 GB. You can use `imread` or the `Tiff` object.

Reading of XLSM, XLTX, and XLTM files on all platforms with xlsread function

The `xlsread` function now reads data from XLSM, XLTX, and XLTM files on all platforms. Previously, this functionality was available only on Microsoft Windows systems with Excel software.

xlsread function now supporting named ranges on all platforms

The `xlsread` function now supports named ranges as inputs, on all platforms. For example, you can call

```
num = xlsread(filename,sheet,xlRange);
```

where `xlRange` refers to a named range of data in your Excel workbook. Previously, this functionality was available only on Microsoft Windows systems with Excel software.

Multiple delimiter support in textscan function

The `textscan` function recognizes multiple different delimiters within a single file. `textscan` also can treat multiple repeated delimiter characters as a single delimiter.

Timeout, user agent, and character encoding inputs for urlread and urlwrite functions

The `urlread` and `urlwrite` functions accept the following optional attribute-value pair arguments.

Attribute	Value
`Timeout`	Timeout duration
`UserAgent`	Client user agent identification
`Charset`	Character encoding, determined from the headers of the file

For example, to download Web content from a the MATLAB Central File Exchange while specifying a timeout duration of 5 seconds, type

```
urlread('http://www.mathworks.com/matlabcentral/fileexchange','Timeout',5);
```

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`avifile`	Warns	`VideoWriter`	Replace all instances of `avifile` with `VideoWriter`.
`wavplay`	Warns	`audioplayer` and `play`	Replace all existing instances of `wavplay` with `audioplayer` and `play`.
`wavrecord`	Warns	`audiorecorder` and `record`	Replace all existing instances of `wavrecord` with `audiorecorder` and `record`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`wk1finfo`	Errors		Remove all instances of `wk1finfo`. Get information about Excel spreadsheets with `xlsfinfo`.
`wk1read`	Errors		Remove all instances of `wk1read`. Read Excel spreadsheets with `xlsread`.
`wk1write`	Errors		Remove all instances of `wk1write`. Write to Excel spreadsheets with `xlswrite`.

R2012a

Desktop Tools and Development Environment

Transpose and Sort Variables in the Variable Editor

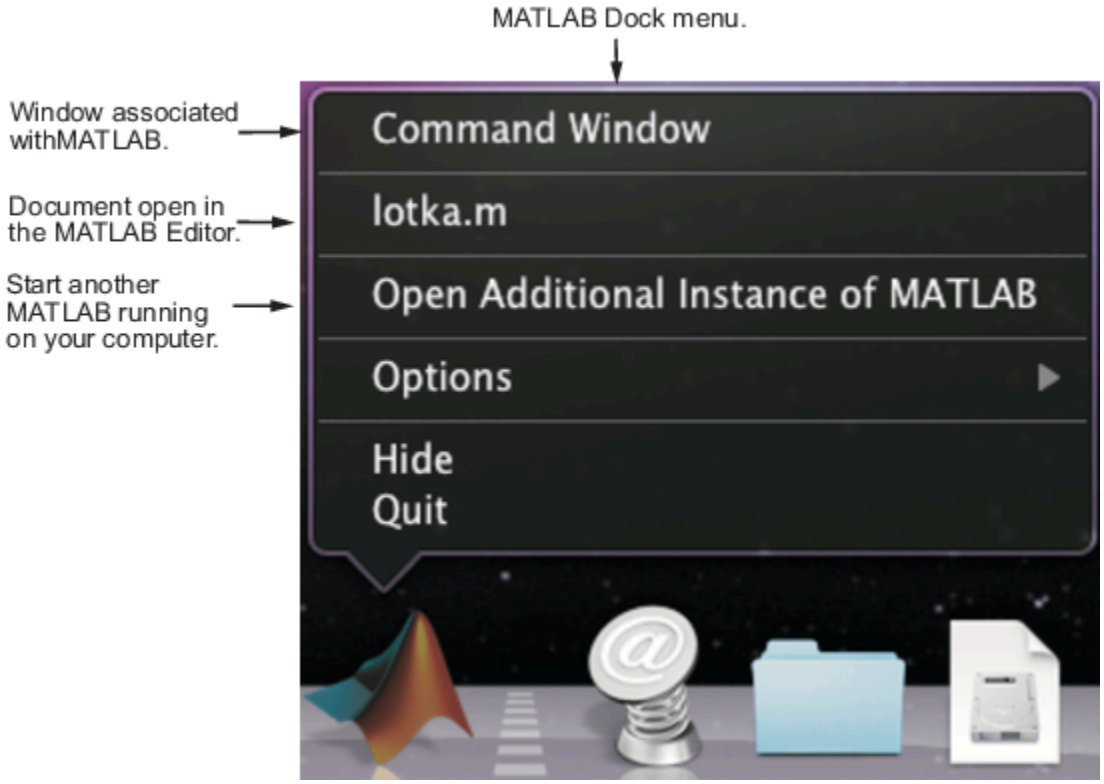
To transpose or sort variables, use the right-click menu options in the Variable Editor. You can sort variables based on single or multiple column selections. The columns you select can be noncontiguous and should always include all the rows. Also, you can create variables from noncontiguous data selections. For more information see the New Variable Editor Features in Release 2012a demo.

MATLAB Dock Menu on Mac Includes New Capabilities

Previously, on a Mac, the Dock menu associated with the MATLAB icon included only the default options, such as **Hide/Show** and **Quit**. Now, consistent with the behavior of most Mac applications, the MATLAB Dock menu lists open windows and documents associated with the running MATLAB application. By choosing on any of the entries in the Dock menu, you can bring that window or document to the front. To view the Dock menu, right-click the MATLAB icon in the Dock or **Ctrl+**click the icon, if you have a one-button mouse.

In addition, the MATLAB Dock menu includes a new option, named **Open Additional Instance of MATLAB**, that is a convenient way to start another instance of MATLAB running on your computer.

The following figure shows the MATLAB Dock menu with a document open in the MATLAB Editor.



Improved Rendering in MATLAB Web Browser

The MATLAB Web browser uses a new HTML rendering engine on Microsoft Windows systems. This new engine provides better rendering, particularly on 64-bit systems. This update impacts the display of HTML from these sources:

- `web` commands
- Published MATLAB file output
- Folder reports from the Current Folder browser
- Simulink Model Advisor

Technical Support Requests Use Proxy Settings

You can submit Technical Support requests from MATLAB by selecting **Help > Submit a MathWorks Support Request**. In previous releases, if you were connected to the Internet via a proxy server, you had to save the request form to a text file, and then email it to MathWorks. Starting with R2012a, you can submit the request directly from the form.

Published Code Can Display Syntax Highlighted Sample Code

When you publish MATLAB code, you can have sample code appear with syntax highlighting. Sample code is code that appears within comments. Previously, only uncommented code appeared with syntax highlighting.

For details, see Syntax Highlighted Sample Code. For a video demo, see Syntax Highlighting for Commented MATLAB Code in Published Scripts in Release 2012a.

Compatibility Considerations

Previously, to publish preformatted text (without syntax highlighting), MATLAB required two or more spaces between the comment character (%) and the first text character in a code cell comment block. Now, to publish such text, there must be two, four, or more spaces in that location. Three spaces results in syntax highlighted sample code.

The publish Function Accepts Name-Value Pairs

You can customize the output from the `publish` function using Name-Value pair arguments. Previously, to customize the output from this function, you had to use a structure. You now can use either a structure or Name-Value pair arguments. For details, see `publish`.

Internationalization

Displaying Non-7-Bit ASCII Characters on Mac OS X Platforms

On Apple Mac OS X platforms, you might see garbled text if you try to use non-7-bit ASCII characters. For example, if you create a variable in the MATLAB command window containing these characters, you can type the characters, but if you display the contents of the variable, the characters are corrupted.

Or if you share MATLAB text files with a user on a computer with different locale settings, they might see corrupted text. The characters are rendered in that user's locale, not the language it was written in.

MATLAB text files include scripts and user-defined functions and classes. MATLAB displays characters in a text file using an encoding scheme, which is defined in the MathWorks locale database. This encoding scheme supports characters for the language specified by the user locale setting. If you try to use non-7-bit ASCII characters in a text file, you might have to use a different encoding scheme. UTF-8 is often used to handle multilingual characters. Beginning in R2011b, to handle non-7-bit ASCII characters, you can change the default encoding scheme to UTF-8 by switching the MathWorks locale database. Note that changing the default encoding scheme might cause characters other than 7-bit ASCII characters in existing text files to be garbled.

See How the MATLAB Process Uses Locale Settings for more information.

How to Change MATLAB's Default Encoding to UTF-8

If you have text files containing non-7-bit ASCII characters, you must convert the encoding in the files before changing the default encoding on your computer. For instructions, see "How to Convert Text File Encoding to UTF-8" on page 18-4.

You can change the default encoding scheme on Mac OS X platforms by using the UTF-8 locale database found in the *`matlabroot`*`/bin` folder.

To change the default locale database, type:

```
mldir = fullfile(matlabroot,'bin');
copyfile(fullfile(mldir,'lcdata.xml'),...
  fullfile(mldir,'lcdata_default.xml'));
copyfile(fullfile(mldir,'lcdata_utf8.xml'),...
  fullfile(mldir,'lcdata.xml'));
```

Alternatively, you can change the default locale database by setting an environment variable on your computer.

1. Set the environment variable, `MWLOCALE_LCDATA_FILENAME`, with the UTF-8 version of the locale database name, `lcdata_utf8.xml`, in the environment file, `environment.plist`.
2. Close MATLAB if it is currently running.
3. Start MATLAB by double-clicking the MATLAB icon in the Applications folder.

How to Convert Text File Encoding to UTF-8

Before converting a file, copy the file to a new directory.

Use the Mac OS X TextEdit application to convert the file encoding. For example:

1. Open a MATLAB text file with TextEdit.
2. Select **File** -> **Save as...**
3. Change the file name.
4. Change **Plain Text Encoding:** to `Unicode (UTF-8)`.
5. Save the file.

Alternatively, the following MATLAB function, `convert_file_encoding`, creates a new text file with different encoding.

```
function convert_file_encoding(infile,outfile,from_encoding,to_encoding)

if strcmp(infile(end-2:end),'mdl');
    isMDL = 1;
else
    isMDL = 0;
end


fpIn = fopen(infile,'r','n',from_encoding);
fpOut = fopen(outfile,'w','n',to_encoding);

while feof(fpIn) == 0
    lineIn = fgets(fpIn);
    if isMDL && strncmp('SavedCharacterEncoding',strtrim(lineIn),22)
        lineIn = regexprep(lineIn,from_encoding,to_encoding);
    end

    fwrite(fpOut,lineIn,'char');
end

fclose(fpIn);
fclose(fpOut);
end
```

To use this function, you need to know the current encoding, `from_encoding`.

- For MATLAB R2010b or later:

  ```
  ret = feature('locale');
  from_encoding = ret.encoding;
  ```

- For MATLAB R2008a through R2010a:

  ```
  ret = feature('locale');
  [t,r] = strtok(ret.ctype,'.');
  from_encoding = r(2:end);
  ```

For example, a file, `myFile.m`, was created with MATLAB encoding set to `ISO-8859-1`. To convert the file to `UTF-8`, type:

```
convert_file_encoding('myFile.m','myFileUTF8.m','ISO-8859-1','UTF-8')
```

Compatibility Considerations

In a future release, the UTF-8 version of the MathWorks locale database will be the default on Mac OS X systems. Any text file created in MATLAB that is encoded in the user default encoding and contains characters other than 7-bit ASCII characters must be converted to the UTF-8 encoding scheme. Otherwise, MATLAB and other MathWorks products might not be able to properly load the files.

You must convert the file encoding before changing the default locale database.

Do not convert text files to UTF-8 if you share them with users on Windows platforms.

Mathematics

New Integral Functions

The new functions, `integral`, `integral2`, and `integral3` perform numerical integration with additional support for nonrectangular and unbounded regions of integration. They are the recommended functions for performing quadrature.

Performance Enhancements

The following functions show improved performance:

- Arithmetic and similar basic math functions for double, single, and integer data types.
- The grid-based interpolation functions, `interp2`, `interp3`, and `interpn`.
- Generating random values using either `rng('combRecursive')` or `RandStream('mrg32k3a')`.

griddata Supports 3-D Data and Natural Neighbor Interpolation

`griddata` is now the recommended function for interpolating 2-D and 3-D scattered data. `griddata` also has a new `method` option, `'natural'`, for specifying natural neighbor interpolation.

TriScatteredInterp Accepts Complex Values

You can now use `TriScatteredInterp` to interpolate complex scattered data in a single pass. For example, `F = TriScatteredInterp(X,V)` now accepts a complex vector `V`.

Set Functions Provide Option to Return Sets in Original Order

You now can specify the ordering of the output array returned by the functions `unique`, `union`, `intersect`, `setdiff`, and `setxor`. The new argument, `setOrder`, is one of two strings, `'stable'` or `'sorted'`. Specify `'stable'` if you want the elements in the output array to be in the same order as in the input array. For example:

`C = unique([9 2 2],'stable')` returns `C = [9 2]`.

Specify `'sorted'` if you want the elements in the output array to be in sorted order. For example:

`C = unique([9 2 2],'sorted')` returns `C = [2 9]`.

Set Functions Changing Behavior in a Future Release

In a future release, the behavior of `unique`, `union`, `intersect`, `setdiff`, `setxor`, and `ismember` will change.

- If there are repeated elements in the input arrays, the functions `unique`, `union`, `intersect`, `setdiff`, and `setxor` will return the index to the first occurrence of the repeated elements (or rows).

- The optional output array, `locb`, returned by `ismember`, will contain the lowest absolute indices in `B` for elements (or rows) that are also in `A`.
- All index vectors returned by `unique`, `union`, `intersect`, `setdiff`, or `setxor` will be column vectors.
- `unique` and `union` will support objects with methods `sort` (`sortrows` for the `'rows'` option), and `ne`. This includes heterogeneous arrays derived from the same root class.
- `intersect`, `setdiff`, `setxor`, and `ismember` will support objects with methods `sort` (`sortrows` for the `'rows'` option), `ne`, and `eq`. This includes heterogeneous arrays derived from the same root class.
- The input arrays passed to `union`, `intersect`, `setdiff`, `setxor`, and `ismember` must be of the same class with the following exceptions:
  - Logical, char, and all numeric classes can combine with double arrays.
  - Cell arrays of strings can combine with char arrays.

Note The set functions already conform to this behavior when you call them with the new `setOrder` argument on page 18-6.

Compatibility Considerations

The new behavior change is introduced for adoption in R2012a. If you want to assess the impact this change might have on your existing code, specify `'R2012a'` as the final input argument to `unique`, `union`, `intersect`, `setdiff`, `setxor`, and `ismember`. For example:

```
[C,IA,IC] = unique([9 9 1],'R2012a')

C =

     1     9


IA =

     3
     1


IC =

     2
     2
     1
```

If the changes adversely affect your code, you can specify `'legacy'` to preserve the current behavior. For example:

```
[C,IA,IC] = unique([9 9 1],'legacy')

C =

     1     9
```

```
IA =

     3     2

IC =

     2     2     1
```

Interpolation and Computational Geometry Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`interp1q`	Still Runs	`interp1`	Replace all instances of `interp1q` with `interp1`.
`interp1(..., 'cubic')`	Still Runs	`interp1(..., 'pchip')`	Replace all instances of `interp1(..., 'cubic')` with `interp1(..., 'pchip')`.

Passing nonuniformly spaced points or grid to:
`interp1(..., 'v5cubic')`
`interp2(..., 'cubic')`
`interp3(..., 'cubic')`
`interpn(..., 'cubic')`
	Warns	`interp1(..., 'spline')`
		`interp2(..., 'spline')`
		`interp3(..., 'spline')`
		`interpn(..., 'spline')`
			In previous releases, `interp1`, `interp2`, `interp3`, and `interpn` silently changed the `'v5cubic'` and `'cubic'` methods to `'spline'` when the sample data was not uniformly spaced. Now, a warning is issued if the uniformity conditions are not honored. To avoid the warning message, change any instances that call for`'v5cubic'` or `'cubic'` to the `'spline'` method.

Passing the `'pp'` flag to `interp1`. For example:
`pp = interp1(x, v, 'linear', 'pp');`
`vq = ppval(pp, xq);`
	Still Runs	Use `griddedInterpolant` to create an interpolating function that is efficient to evaluate in a repeated manner.	Replace all instances of `interp1(..., 'pp')` with `griddedInterpolant`. For example:
			`F = griddedInterpolant(x, v, 'linear');`
			`vq = F(xq);`

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`interp1(X, Y, Xq, [], ...)` where `[]` specifies the default interpolation method	Still Runs	`interp1(X, Y, Xq, 'linear', ...)`	Replace all instances of `interp1(X, Y, Xq, [], ...)` with `interp1(X, Y, Xq, 'linear', ...)`.

`interp1(x,V,..)` where `x` is a vector and `V` is an array representing multiple value sets. For example:
`x = (1:5)';`
`V = [x, 2*x, 3*x];`
`xq = (1:0.5:5)';`
`Vq = interp1(x, V, xq);`
	Still Runs	Use `griddedInterpolant` and loop over each value set, updating and evaluating during each iteration.	Replace all instances of `interp1(x,V,...)`, where `x` is a vector and `V` is an array representing multiple value sets. Loop over `V`, calling `griddedInterpolant` on each value set. The following example code shows how to set up and interpolate over three value sets in `V` :
			`x = (1:5)';`
			`V = [x, 2*x, 3*x];`
			`xq = (1:0.5:5)';`
			`F = griddedInterpolant(x, V(:,1));`
			`for n=1:3`
			`  F.Values = V(:, n);`
			`  Vq(:, n) = F(xq);`
			`end`
			The columns of `Vq` are the interpolation results for the corresponding columns of `V`.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed orientation vectors to `interp2`:
`Vq = interp2(x, y, V, xq, yq)`
Specifically, if one or both of the following are true:
• One of `x` and `y` is a row vector and the other is a column vector.
• One of `xq` and `yq` is a row vector and the other is a column vector.
	Still Runs	Construct the full grid with `meshgrid` first. Alternatively, use `griddedInterpolant` if you have a large data set.	Modify all instances that pass mixed orientation vectors to `interp2`. You can modify your code in one of two ways:
			• Call `meshgrid` to construct the full grid first.
			`[X,Y] = meshgrid(x, y);`
			`[Xq,Yq] = meshgrid(xq, yq);`
			`Vq = interp2(X, Y, V, Xq, Yq);`
			• Pass the vectors to `griddedInterpolant` inside a cell array.
			`F = griddedInterpolant({x, y}, V.');`
			`Vq = (F({xq, yq})).'`

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed orientation vectors to `interp3`: `interp3(x, y, z, V, xq, yq, zq)`
Specifically, if one or both of the following are true:
• `x`, `y`, and `z` are a combination of row and column vectors.
• `xq`, `yq`, and `zq` are a combination of row and column vectors.
	Still Runs	Construct the full grid with `meshgrid` first. Alternatively, use `griddedInterpolant` if you have a large data set.	Modify all instances that pass mixed orientation vectors to `interp3`. You can modify your code in one of two ways:
			• Call `meshgrid` to construct the full grid first.
			`[X,Y,Z] = meshgrid(x, y, z);`
			`[Xq,Yq,Zq] = meshgrid(xq, yq, zq);`
			`Vq = interp3(X, Y, Z, V, Xq, Yq, Zq);`
			• Pass the vectors to `griddedInterpolant` inside a cell array.
			`V = permute(V, [2 1 3]);`
			`F = griddedInterpolant({x, y, z}, V);`
			`Vq = F({xq, yq, zq});`
			`Vq = permute(Vq, [2 1 3]);`

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed orientation vectors to `interpn`: `interpn(x1,x2,...,xn, V, x1q,x2q,...,xnq)` Specifically, if one or both of the following are true:
• `x1,x2,...,xn` are a combination of row and column vectors.
• `x1q,x2q,...,xnq` are a combination of row and column vectors.
	Still Runs	Construct the full grid with `ndgrid` first. Alternatively, use `griddedInterpolant` if you have a large data set.	Modify all instances that pass mixed orientation vectors to `interpn`. You can modify your code in one of two ways:
			• Call `ndgrid` to construct the full grid first.
			`[X1,X2,...,Xn]= ndgrid(x1,x2,...,xn) ; [X1q,X2q,...,Xnq]= ndgrid(x1q,x2q,...,xnq); Vq= interpn(X1,X2,...,Xn, V, X1q,X2q,...,Xnq);`
			• Pass the vectors to `griddedInterpolant` inside a cell array.
			`F= griddedInterpolant({x1, x2,...,xn}, V); Vq= F({x1q, x2q,...,xnq});`

`interp1(..., *METHOD)` `interp2(..., *METHOD)` `interp3(..., *METHOD)` `interpn(..., *METHOD)`	Still Runs Errors for invalid input data	`interp1(..., METHOD)` `interp2(..., METHOD)` `interp3(..., METHOD)` `interpn(..., METHOD)`	In previous releases, `interp1`,`interp2`,`interp3`, and `interpn` provided a `*METHOD` option that bypassed error checking on the assumption of valid data. These checks are no longer bypassed. Use the `griddedInterpolant` class to perform repeated interpolation queries on the same data set without penalty of repeated error checks.
`vq= griddata(x,y,v,xq,yq)` where `xq` is a row vector and `yq` is a column vector.	Still Runs	`vq= griddata(x,y,v,Xq,Yq)` where `Xq` and `Yq` are the output arrays returned by `meshgrid`.	Modify all instances that pass mixed orientation vectors to `griddata`. To specify a grid of query points, construct a full grid with `meshgrid` before calling `griddata`.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`griddata3`	Errors	`griddata`	Replace all existing instances of `griddata3` with `griddata`.
`delaunay3`	Errors	`delaunay`	Replace all existing instances of `delaunay3` with `delaunay`.
`tsearch`	Errors	`DelaunayTri/pointLocation`	Replace all existing instances of `tsearch` with `DelaunayTri/pointLocation`.
`dsearch`	Errors	`DelaunayTri/nearestNeighbor`	Replace all existing instances of `dsearch` with `DelaunayTri/nearestNeighbor`.
`convhull(..., OPTIONS)`	Errors	Omit the `OPTIONS` argument when you call `convhull`.	Remove the `OPTIONS` argument from all instances that pass it to `convhull`.
`delaunay(..., OPTIONS)`	Errors	Omit the `OPTIONS` argument when you call `delaunay`.	Remove the `OPTIONS` argument from all instances that pass it to `delaunay`.
`griddata(..., OPTIONS)`	Errors	Omit the `OPTIONS` argument when you call `griddata`.	Remove the `OPTIONS` argument from all instances that pass it to `griddata`.
`voronoi(..., OPTIONS)`	Errors	Omit the `OPTIONS` argument when you call `voronoi`.	Remove the `OPTIONS` argument from all instances that pass it to `voronoi`.

Other Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`isequalwithequalnans`	Still Runs	`isequaln`	Replace all instances of `isequalwithequalnans` with `isequaln`.
`mimofr`	Still Runs	Not Applicable	Remove all instances of `mimofr` from your existing code.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`RandnAlg` property of `RandStream` class	Warns	`NormalTransform` property of `RandStream` class	Replace all existing instances of `RandnAlg` with `NormalTransform`.
`setDefaultStream` method of `RandStream` class	Warns	`setGlobalStream` method of `RandStream` class	Replace all existing instances of `RandStream.setDefaultStream` with `RandStream.setGlobalStream`.
`getDefaultStream` method of `RandStream` class	Warns	`getGlobalStream` method of `RandStream` class	Replace all existing instances of `RandStream.getDefaultStream` with `RandStream.getGlobalStream`.
`atan2(y,x)` for complex `y` and `x`.	Errors	`atan2(real(y),real(x))` when `y` and `x` are complex.	Replace all existing instances of `atan2(y,x)` with `atan2(real(y),real(x))` where `y` and `x` are complex.
Second output argument for `besselh`, `besseli`, `besselj`, `besselk`, `bessely`, and `airy`. For example, `[J,ierr] = besselj(nu,Z)`.	Warns	Syntax that returns only the solution vector. For example, `J = besselj(nu,Z)`.	Replace all instances that return two output arguments with the syntax that returns only the solution vector.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing mixed orientation input vectors to `besselh`, `besseli`, `besselj`, `besselk`, and `bessely`. For example, passing a row vector followed by a column vector:
`J= besselj(rowNu, colZ)`
or passing a column vector followed by a row vector:
`J= besselj(colNu, rowZ)`
	Warns	Construct the inputs with `ndgrid` or `meshgrid` first. Alternatively, you can pass a function handle and the mixed orientation vectors to `bsxfun`.	Modify all instances that pass mixed orientation vectors. You can modify your code in one of two ways:
			• Call `meshgrid` or `ndgrid` to construct the full grid first.
			`[nu,Z]= meshgrid(rowNu, ColZ);`
			`J= besselj(nu, Z);`
			• Pass a function handle and the mixed orientation vectors to `bsxfun`. For example, if your existing code passes a row vector followed by a column vector, make the following change:
			`J= bsxfun(@besselj, rowNu, colZ);`or, if your code passes a column vector followed by a row vector, make the following change:
			`J= bsxfun(@besselj, colNu', rowZ.');`

`Y = psi(k0:k1,X)`	Warns	`Y = psi(k,X)` where `k` is a scalar specifying the `k`th derivative of ψ at the elements of `X`.	Replace all instances of `Y = psi(k0:k1,X)` with `Y = psi(k,X)`, where `k` is a scalar. To modify your code, loop through the values `k0:k1`. For example:
			`for k=k0:k1`
			`  Y(:,k) = psi(k,X);`
			`end`
			In the future, `size(Y)` will be `size(X)`. Modify any code that depends on `size(Y)`.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations

Passing empty and nonscalar input to `besselh`, `besseli`, `besselj`, `besselk`, `bessely`, and `airy`. For example,
`J = besselj([],(1:3))`
or
`J = besselj((1:3),[])`
	Warns	`J = besselj(nu,[])` or `J = besselj([],Z)` where `nu` and `Z` are scalars.	Modify all instances that pass combinations of empty arrays with nonscalar input. The inputs must be the same size or one must be a scalar.

Data Analysis

Programming

xlsread Reads XLSX Files on All Platforms

The `xlsread` function now reads data from XLSX files on all platforms, including support for specifying the range and worksheet number. Previously, this functionality was available only on Microsoft Windows systems with Excel software.

VideoWriter Supports MPEG-4 Files on Windows 7 Systems

`VideoWriter` now creates MPEG-4 files with the extension `.mp4` or `.m4v` on Windows 7 systems. For more information, see the `VideoWriter` reference page.

audioplayer Supports Overlapping Playback

In R2011a, changes that allowed `audioplayer` to support device selection also disabled the ability to play overlapping audio samples. With R2012a, overlapping playback is available again.

For example, play a second sample before the first one completes, and hear audio from both:

```
chirpData = load('chirp.mat');
chirpObj = audioplayer(chirpData.y,chirpData.Fs);

gongData = load('gong.mat');
gongObj = audioplayer(gongData.y,gongData.Fs);

play(chirpObj);
play(gongObj);
```

importdata Returns Different Output for Some Text Files

Previously, the `importdata` function recognized comments in text files and did not include them in the output. In addition, `importdata` parsed files that contained numeric values and comma, space, or tab delimiters into a numeric array, even when you specified a delimiter that did not match the contents of the file.

In this release, `importdata` honors the specified delimiter, and includes comments in the output. For example, consider a hypothetical file named `test.txt` with this space-delimited data:

```
% Comment in file
1 2 3
4 5 6
```

In previous releases,

```
t = importdata('test.txt',',')
```

returned a numeric array:

```
t =
     1     2     3
     4     5     6
```

In this release, the same code returns a cell array:

```
t =
    '% Comment in file'
    '1 2 3'
    '4 5 6'
```

If you remove the incorrect comma delimiter,

```
t = importdata('test.txt')
```

`importdata` returns a struct array:

```
t =
        data: [2x3 double]
    textdata: {'% Comment in file'}
```

Exponents Print with Two Digits

In previous releases, functions that printed floating-point values with exponents used three digits for the exponent on Windows systems, but two digits on any other system. Now, these functions use two digits for the exponent on all systems.

For example,

```
str = sprintf('%e',pi)
```

always returns

```
str =
3.141593e+00
```

Conversion of Error and Warning Message Identifiers

For R2012a, error and warning message identifiers have changed in MATLAB.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `MATLAB:eigs:NonPosIntSize` identifier has changed to `MATLAB:eigs:RoundNonIntSize`. If your code checks for `MATLAB:eigs:NonPosIntSize`, you must update it to check for `MATLAB:eigs:RoundNonIntSize` instead. For a mapping of the new identifiers to the original identifiers, see Technical Support solution `1-ERAFNC`.

Specify a List of Allowed Subclasses in the Class Definition

You can control which classes are allowed to subclass a class that you define by specifying a list classes in the new class attribute, `AllowedSubclasses`. See Controlling Allowed Subclasses for more information.

Specify Which Classes Can Access Class Members

You can control access to specific class members (properties, methods, and events) by specifying a list of classes to which you want to grant access to these members. See Controlling Access to Class Members for more information.

Compatibility Considerations

In previous releases, you could assign a value to the following class member access attributes as a value returned by a function:

- Properties – `Access`, `GetAccess`, and `SetAccess`
- Methods – `Access`
- Events – `ListenAccess` and `NotifyAccess`

With this release, you can specify values for these attributes only explicitly as either the appropriate character string or a `meta.class` object returned by the `?` operator, or a cell array of character strings and/or `meta.class` objects. See Specify Access to Class Members for more information.

Method Declared as Abstract and Private Now Errors

In previously releases, declaring a method as both `Abstract` and `Access = private` did not cause an error. With this release, declaring a method both `Abstract` and `private` causes an error.

Compatibility Considerations

If any of your class definitions declare a method as both `Abstract` and `private`, you need to declare the method either as `Abstract`, which make the class abstract and requires subclasses to implement the method with the declared syntax, or declare the method `Access = private`, which means that only methods within the defining class can call the `private` method.

Declaring a method as both `Abstract` and `private` is not supported.

New Capabilities for Writing Image Data to FITS Files

MATLAB now includes a function named `fitswrite` that you can use to write image data to Flexible Image Transport System (FITS) files. You can also display the metadata for all Header-Data Units (HDUs) in a FITS file using the `fitsdisp` function. (MATLAB already supports reading FITS files using the `fitsread` function.)

In addition, MATLAB now includes a new package, called `matlab.io.fits`, containing dozens of functions that provide access to the routines in the CFITSIO library. Using these low-level functions, you can create FITS files, read data from FITS files and write data to the files, using the CFITSIO library, version 3.27.

Access Data on Remote Servers Using the OPeNDAP Protocol

You can read data stored on remote servers using the OPeNDAP protocol capability of the following MATLAB NetCDF functions:

- `ncread`
- `ncreadatt`
- `ncdisp`
- `ncinfo`
- `netcdf.open`

To use these functions to access data on remote servers using OPeNDAP protocol, specify the URL instead of a file name.

Upgrades to Scientific File Format Libraries

The following table lists upgrades to scientific file format libraries used by MATLAB.

Library	Version
NetCDF	4.1.3 (upgraded from 4.1.2)

Ability to Read NetCDF Files Using HDF4 Functions Removed

In past releases, for certain NetCDF files on Linux and Macintosh systems, you could read the files using the HDF4 functions. This capability has been removed. To read NetCDF files, use the high-level NetCDF functions or the low-level NetCDF package.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`avifile`	Still runs	`VideoWriter`	Replace all instances of `avifile` with `VideoWriter`.
`fileparts` return argument 4 (file version)	Errors — beginning in MATLAB Version 7.13 (R2011b)	No alternative; file versions are unsupported	Call `fileparts` with three return arguments: `[path_name, file_name, file_extension]`
`helpbrowser`	Still runs	`doc`	Replace all instances of `helpbrowser` with `doc`.
`helpdesk`	Still runs	`doc`	Replace all instances of `helpdesk` with `doc`.
`helpwin`	Still runs	`doc`	Replace all instances of `helpwin` with `doc`.
`info`	Warns		Remove all instances of `info`. Find information about MathWorks at `www.mathworks.com/company/aboutus/contact_us`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`mmreader`	Warns, creates a `VideoReader` object	`VideoReader`	Replace all instances of `mmreader` with `VideoReader`.
`mmreader.isPlatformSupported` and `VideoReader.isPlatformSupported`	Errors		For a platform-specific list of supported video file formats, use `getFileFormats`.
`support`	Warns		Remove all instances of `support`. Find the MathWorks technical support page at `www.mathworks.com/support`.
`whatsnew`	Warns		Remove all instances of `whatsnew`.

Graphics and 3–D Visualization

Creating Graphical User Interfaces (GUIs)

External Interfaces/API

Changes to Compiler Support

New Compiler Support

MATLAB Version 7.14 (R2012a) supports these new compilers for building MEX-files on Linux 64- and 32-bit platforms:

- GNU gcc Version 4.4.6

MATLAB supports these new compilers for building MEX-files on Windows 64- and 32-bit platforms:

- Intel Visual Fortran Composer XE 2011 SP1 (12.1)

Discontinued Compiler Support

MATLAB no longer supports the following compilers:

Windows 64-Bit Platforms

- Microsoft Visual Studio 2010 (10.0) Express

Windows 32-Bit Platforms

- Microsoft Visual Studio 2010 (10.0) Express
- Visual C++ 6.0

Linux (64- and 32-Bit) Platforms

- GNU gcc Version 4.3.x

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

mxAssert and mxAssertS Functions Throw MATLAB Exception

The `mxAssert` and `mxAssertS` functions now throw the MATLAB exception `MATLAB:mex:assertion` and terminate the MEX-file instead of aborting MATLAB.

Version Support for COM ProgID Values

You can specify version-specific Programmatic Identifiers (ProgID) for MATLAB Version 7.14 (R2012a) and later.

- `MATLAB.Desktop.Application`
- `MATLAB.Application`
- `MATLAB.Application.Single`, to specify a dedicated server.

R2011b

Desktop Tools and Development Environment

Command Window

Error Messages Reformatted for Improved Readability and Navigation

Error messages in the Command Window no longer appear with question marks or arrows. In addition, error messages for each function now provide a link to that function's documentation.

The following images show examples of the differences in presentation:

- Built-in functions:
  - MATLAB Version 7.13 (R2011b)

    ```
    >> eig
    Error using eig
    Not enough input arguments.
    ```

  - MATLAB Version 7.12 (R2011a) and earlier

    ```
    >> eig
    ??? Error using ==> eig
    Not enough input arguments.
    ```

- Functions that are not built in:
  - MATLAB Version 7.13 (R2011b)

    ```
    >> num2str
    Error using num2str (line 40)
    Not enough input arguments.
    ```

  - MATLAB Version 7.12 (R2011a) and earlier

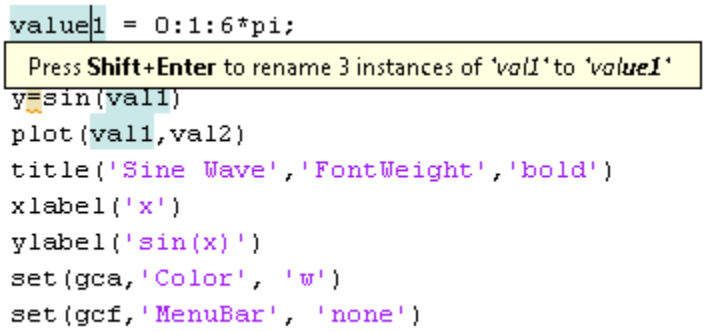
    ```
    >> num2str
    ??? Error using ==> num2str at 42
    Numeric array is unspecified
    ```

Editor

New Desktop features and changes introduced in this version are:

Automatically Renaming All Variables and Functions in File

Under certain conditions, as you rename a variable or function, a tooltip opens. The tooltip indicates that MATLAB can rename all instances of the function or variable in the current file. This is helpful in preventing errors that result if you change names manually and overlook or mistype one or more instances of the name. For a video overview, watch this video demo



For details, including the circumstances under which MATLAB prompts you, see Automatically Rename All Functions or Variables in a File.

Internationalization

Displaying Multilingual Characters on Mac OS X Platforms

On Apple Mac OS X platforms, you might see garbled text if you try to use multilingual characters. Suppose you create a variable in the MATLAB command window containing multilingual characters. You can type the multilingual characters, but if you display the contents of the variable, the characters are corrupted.

MATLAB text files include scripts and user-defined functions and classes. MATLAB displays characters in a text file using an encoding scheme, which is defined in the MathWorks locale database. This encoding scheme supports characters for the language specified by the user locale setting. If you try to use multilingual characters in a text file, you may have to use a different encoding scheme. UTF-8 is often used to handle multilingual characters. In R2011b, to handle multilingual characters, you can change the default encoding scheme to UTF-8 by switching the MathWorks locale database. Note that changing the default encoding scheme might cause characters other than 7-bit ASCII characters in existing text files to be garbled.

See How the MATLAB Process Uses Locale Settings for more information.

How to Change MATLAB's Default Encoding to UTF-8

You can change the default encoding scheme on Mac OS X platforms by using the UTF-8 locale database found in the *`matlabroot`*`/bin` folder. If you have text files containing characters other than 7-bit ASCII characters, you must convert the encoding before changing the default encoding. For instructions, see “How to Convert Text File Encoding to UTF-8” on page 19-3.

To change the default locale database, type:

```
mldir = fullfile(matlabroot,'bin');
copyfile(fullfile(mldir,'lcdata.xml'),...
  fullfile(mldir,'lcdata_default.xml'));
copyfile(fullfile(mldir,'lcdata_utf8.xml'),...
  fullfile(mldir,'lcdata.xml'));
```

How to Convert Text File Encoding to UTF-8

Before converting a file, copy the file to a new directory.

Use the Mac OS X TextEdit application to convert the file encoding. For example:

1. Open a MATLAB text file with TextEdit.
2. Select **File ->Save as...**
3. Change the file name.
4. Change **Plain Text Encoding:** to `Unicode (UTF-8)`.
5. Save the file.

Alternatively, the following MATLAB code creates a new text file with the encoding set to `UTF-8`:

```
function convertencoding(infile, outfile)

fpIn = fopen(infile, 'r', 'n');
fpOut = fopen(outfile, 'w', 'n', 'UTF-8');

while feof(fpIn) == 0
  lineIn = fgets(fpIn);
  fwrite(fpOut, lineIn, 'char');
end

fclose(fpIn);
fclose(fpOut);
end
```

Compatibility Considerations

In a future release, the UTF-8 version of the MathWorks locale database will be the default on Mac OS X systems. Any text file created in MATLAB that is encoded in the user default encoding and contains characters other than 7-bit ASCII characters must be converted to the UTF-8 encoding scheme. Otherwise, MATLAB and other MathWorks products might not be able to properly load the files.

You must convert the file encoding before changing the default locale database.

Mathematics

New Functionality for Grid-Based Interpolation

MATLAB includes a new class, `griddedInterpolant`, for grid-based interpolation. `griddedInterpolant` works with grids in `ndgrid` format and complements the functionality provided by `interpn` to provide improved performance and memory efficiency.

Performance Enhancements

The following functions show improved performance:

- Some integer math functions
- Some linear algebra functions, including `chol` and 3-output form of `qr`
- Trigonometric functions

Permutation Option for randperm

The `randperm` function and the `RandStream.randperm` method now have the option of returning random permutations of `k` integers selected at random from the integers 1 through `n`.

Return Permutation Information in Vector for qr

If you use the three-output form of `qr` in noneconomy mode, you can now specify `'vector'` as an input argument. The permutation information will return a vector instead of a matrix.

Changes to meshgrid and ndgrid

The functions `meshgrid` and `ndgrid` have changed behavior under certain calling scenarios:

- When more than one input argument is passed to `meshgrid`, the dimensionality of the output arrays is deduced from the number of inputs. Previously the dimensionality was deduced from the number of outputs. For example, `[x,y] = meshgrid(1:3,1:4,1:5)` will now assume a missing third output argument and return 3-D output arrays `x` and `y`. Previously, the third input was ignored and the function returned 2-D arrays for `x` and `y`.
- When a single argument is passed to `ndgrid`, the dimensionality of the output arrays is dictated by the number of output arguments. `ndgrid` has been revised to maintain consistency in the case of a single input and output. `ndgrid` will now degenerate naturally to support 1-D outputs. For example, `x = ndgrid(1:5)` returns a 5-by-1 output vector `x`. Previously, a 5-by-5 array was returned.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
`bessel`	Errors	`besselj`	Replace all instances of `bessel` with `besselj`.
ODE solver syntax `solver('vdp',t0,tfinal,y0)` where `solver` is one of the ODE solvers `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, or `ode23tb`	Errors	`solver('vdp',[t0 tfinal],y0)` or `solver(@vdp,[t0 tfinal],y0)`	Replace all instances of the syntax `t0,tfinal` with `[t0 tfinal]`.
Values `on` and `off` for `Mass` property of `odeset` and `odeget`	Errors	Set the `Mass` property to a constant matrix or a function that evaluates the variable mass matrix.	Update your code to use a constant mass matrix or a function that evaluates the variable mass matrix. For more information, check the supported settings of `Mass` and `MassStateDependency` in the `odeset` properties.
`MassConstant` property of `odeset` and `odeget`	Errors	For a constant mass matrix, set the `Mass` property to that matrix. For a time-dependent mass matrix, set the `Mass` property to a function that evaluates the matrix and set the `MassStateDependency` property to `none`.	Update your code to use a constant mass matrix or a function that evaluates the variable mass matrix. For more information, check the supported settings of `Mass` and `MassStateDependency` in the `odeset` properties.

Data Analysis

Programming

Load and Save Parts of Variables in V7.3 MAT-Files

The new `matfile` function creates a `matlab.io.MatFile` object that can efficiently load or save to parts of variables in Version 7.3 MAT-Files. Loading part of a variable requires less memory than loading the entire contents of that variable.

For example, these commands create a MAT-file and add data to part of variable `X`:

```
new = matfile('newfile.mat','Writable',true);
new.X(6:10,6:10) = magic(5);
```

This command loads part of the data into variable `partOfX`:

```
partOfX = new.X(1:8,1:8);
```

Note Syntax such as `size(new.X)` or `new.X(end,end)` temporarily loads the entire contents of variable `X` into memory. To determine the size of a variable without loading, use the `size` method for `matlab.io.MatFile` objects:

```
sizeOfX = size(new,'X');
```

For more information, see the `matfile` reference page or watch this video demo.

Nonmatching Function Name Warning is Now an Error

In previous releases of MATLAB, entering a function name that did not match any known function name in the given letter case resulted in a warning. After displaying this warning message, MATLAB attempted find a function that would match if letter case were ignored. If such a match were found, MATLAB ran this function instead.

In MATLAB R2011b, the MATLAB software generates an error if it can’t find an exact-case match. The message displayed by the error may suggest the correct spelling for the function that was entered. The message identifier for this error reads as follows:

```
'MATLAB:dispatcher:InexactCaseMatch'
```

Compatibility Considerations

If you have a program that calls another program with nonmatching letter case, the call to this function now throws an error.

New narginchk Function Replaces nargchk

`narginchk` is a new function that replaces and adds to the functionality of `nargchk`.

The `narginchk` function differs from `nargchk` in the following ways:

- `narginchk` accepts only 2 inputs: the minimum and maximum allowable number of arguments one can pass to the currently running function.

- `narginchk` returns no outputs. Instead, it throws an error if the number of arguments passed to the current function is outside the allowable range

Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in MATLAB.

Compatibility Considerations

If you have code that uses message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `MATLAB:eigs:NonPosIntSize` identifier has changed to `MATLAB:eigs:RoundNonIntSize`. If your code checks for `MATLAB:eigs:NonPosIntSize`, you must update it to check for `MATLAB:eigs:RoundNonIntSize` instead. For a mapping of the new identifiers to the original identifiers, see Technical Support solution `1-ERAFNC`.

New Spreadsheet Import Tool

The new Spreadsheet Import Tool allows you to select ranges of data and define rules for handling nonnumeric values. This tool opens instead of the Import Wizard for spreadsheets and comma-separated value (CSV) files. For more information, watch this video demo.

Two Functions Added to netCDF Low-Level Package To Aid Performance

The NetCDF low-level package now includes two functions that you can use to improve performance:

- `netcdf.getChunkCache` – Return default chunk cache settings.
- `netcdf.setChunkCache` – Set the default chunk cache settings.

Improved Performance for TIFF Input and Output

Input and output performance for TIFF files has been improved, when using `imread` and `imwrite`.

Upgrades to Scientific File Format Libraries

The following table lists upgrades to scientific file format libraries used by MATLAB.

Library	Version Upgrade
NetCDF	4.0.1 to 4.1.2
HDF5	1.8.3 to 1.8.6
HDF4	4.2r4 to 4.2.5
HDF-EOS	2.16 to 2.17
TIFF	3.7.1 to 3.9.5

VideoReader Supports MPEG-4 and MOV on Windows 7 Systems

`VideoReader` now imports MPEG-4 and Apple QuickTime movies, and any other formats supported by Microsoft Media Foundation, on Windows 7 systems. For more information, see the `VideoReader` reference page.

MATLAB Warns if Listener Defined for Nonobservable Property

If you attempt to create a `PreSet` or `PostSet` listener for a property that is not declared as `SetObservable`, MATLAB issues a `MATLAB:class:nonSetObservableProp` warning.

If you attempt to create a `PreGet` or `PostGet` listener for a property that is not declared as `GetObservable`, MATLAB issues a `MATLAB:class:nonGetObservableProp` warning.

In previous releases, MATLAB did not warn in either case, but did not call the listener callback function when a set or get event occurred. In a future release, these warnings will become errors.

Compatibility Considerations

If you use class metadata or the `properties` function to get a list of properties for which you create listeners, add warning/error handling for these conditions. A better approach is to use `findobj` to get the list of observable properties for any given class and assign listeners only for those properties.

MATLAB Warns if Property Is Not Member of Class When Defining Listener

When calling the `event.proplistener` constructor, or calling the `addlistener` method with an array of `meta.property` objects, warning messages alert you if a property is not a member of the correct class.

- **Nonscalar input objects**: if any one of the properties is not a valid member of the class of the object array, MATLAB issues the `MATLAB:class:PropNotMember` warning .
- **Scalar input object**: if any of the properties are not valid dynamic or class properties, MATLAB issues the `MATLAB:class:DynPropNotMember` or `MATLAB:class:PropNotMember` warning.

In previous releases, MATLAB did not warn in either case. In a future release, these warnings will become errors.

Built-In disp Works with Empty Objects

It is now possible to call the built-in `disp` function with empty objects. For example, define the `TestClass` as:

```
classdef TestClass
   properties
      PropertyA = 1;
   end
end
```

Then call the built-in version of `disp`:

```
>> builtin('disp',TestClass.empty)
  0x0 empty TestClass
```

```
Properties:
  PropertyA

Methods
```

In previous releases, calling the built-in `disp` function did not display anything with empty objects.

MATLAB Warns if Class Defines Property as Dependent and Constant

MATLAB now issues a warning if a class defines properties as both `Dependent` and `Constant`. In a future release, this warning will become an error.

Support for Switch/Case Statements with Objects

You can construct `switch` statement blocks using objects as both the switch expression and the case expression. The object's class must define or inherit an `eq` (==) method. See Using Switch/Case Statements with Objects in Functions Used with Objects for more information.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`nargchk`	Still runs	`narginchk`	Accepts only `min` and `max` inputs, returns no output, generates error.
`nargoutchk`	Still runs	`nargoutchk` (see new syntax)	Accepts only `min` and `max` inputs, returns no output, generates error.

Graphics and 3-D Visualization

Creating Graphical User Interfaces (GUIs)

External Interfaces/API

Changes to Compiler Support

New Compiler Support

MATLAB Version 7.13 (R2011b) supports these new compilers for building MEX-files on Windows 64- and 32-bit platforms:

- Microsoft Windows Software Development Kit for Windows 7 and .NET Framework 4, Version 7.1
- Intel C++ Composer XE 2011
- Intel Visual Fortran Composer XE 2011

MATLAB Version 7.13 (R2011b) supports the following new compiler for building MEX-files on Mac OS X 64-bit platforms:

- Apple Xcode 4.0.0 with gcc 4.2.x

Compiler Support To Be Phased Out

Support for the following compilers on Windows 64- and 32-bit platforms will be discontinued in a future release, at which time new versions will be supported. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

- Intel C++ Version 11.1
- Intel Fortran Version 11.1

New Object Support for mxGetProperty and mxSetProperty Functions

You can read and modify properties of Java and COM objects using the `mxGetProperty` and `mxSetProperty` functions.

MEX Header File Does Not Define C++ Type char16_t

The `matrix.h` header file, used by the `mex.h` header file for building C/C++ MEX-files, no longer defines the C++ type `char16_t`.

Compatibility Considerations

If your MEX-file uses the `char16_t` type, you must include the appropriate header file for your compiler in order to build the MEX-file. For example, the header file for Microsoft Visual Studio Version 10.0 is `yvals.h`.

New Support for Features in Microsoft .NET Framework

- “Support for Cell Arrays” on page 19-15
- “Support for Auto-Conversion of Multidimensional Arrays” on page 19-15

Support for Cell Arrays

Creating Cell Arrays from .NET Strings and Objects

Use the MATLAB `cell` function to convert .NET `System.String` and `System.Object` arrays to MATLAB cell arrays. For more information, see Converting .NET Arrays to Cell Arrays.

Passing Cell Arrays to .NET Methods

If an input argument to a .NET property or method is an array of `System.Object` or `System.String`, you can pass a cell array. MATLAB automatically converts a cell array into the appropriate .NET array. For more information, see Pass Cell Arrays.

Support for Auto-Conversion of Multidimensional Arrays

You can pass MATLAB arrays directly to .NET without explicitly converting them into .NET arrays. MATLAB also converts an array of a lower dimensionality to a higher dimensionality .NET array. For more information, see Pass Arrays.

Compatibility Considerations

In most cases, you no longer need to use the `NET.convertArray` function. Review your MATLAB code to update the use of this function. If you continue to use `NET.convertArray`, note the following change of behavior for empty matrices:

- If you call `NET.convertArray` with an N-D empty matrix and try to convert it to 1-D .NET array, MATLAB throws the following error:

  ```
  Source and destination array dimension sizes do not match.
  ```

- Different results for:

  ```
  NET.convertArray(rand(0,1,1))
  ```

  As of R2011a, MATLAB creates a `System.Double[,]`. In R2011b, MATLAB creates a `System.Double[]`.

COM Automation Server Error Message Formatting

Although some MATLAB Command Window messages have been reformatted (see “Error Messages Reformatted for Improved Readability and Navigation” on page 19-2), error messages from the `Execute` function can still be identified with leading `???` characters.

R2011a

Desktop Tools and Development Environment

Desktop

New features and changes introduced in Version 7.12 (R2011a) are:

MATLAB Menus Display at the Top of the Apple Mac Screen

Previously when running on Apple Mac, menus displayed at the top of the MATLAB desktop. In addition, if a tool was undocked from the desktop, the menu displayed at the top of that tool. Now, to be consistent with the behavior of most Mac applications, MATLAB menus display at the top of the Mac screen.

Help Browser

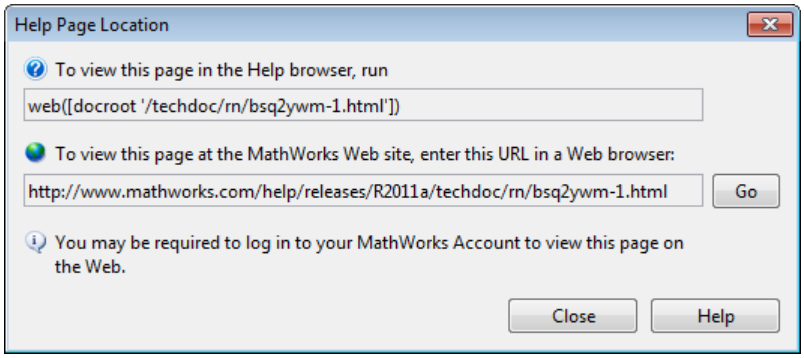
New Location and Archived Content for Product Documentation

The new MathWorks Web site location for product documentation is https://www.mathworks.com/help/. Accessing the old URL, `www.mathworks.com/access/helpdesk/help/helpdesk.html`, redirects you the new location.

In addition to all current product documentation, the new location provides archived product documentation from all releases going back to R13sp2 (including Japanese, when available). To go to the archive, click the View documentation for other releases link. The archived release documentation is available only at the MathWorks Web site, and not via the Help browser.

To view some documentation you must log on to `www.mathworks.com` with your MathWorks user name and password. For example, you need to log in to view current PDF files and documentation from prior releases.

The **View > Page Location** dialog box in the Help browser now links to documentation at the MathWorks Web site that accompanies the release you are currently running. For example, when you run R2011a, the dialog box looks like this.



As of R2011a, the Help Browser **View > Page Location** dialog box (shown above) is available on Japanese language systems.

Submit Support Requests Directly from MATLAB

As of this release, if you need assistance from MathWorks product support, you can request it directly from the MATLAB desktop. Selecting the **Submit a MathWorks Support Request** option from the **Help** menu opens a login dialog for you to provide the e-mail address and password for your MathWorks account. After you log in, a form displays for you to create a service request. Select the product for which you want help, explain the issue you are having, give the request a meaningful title, describe the issue you are having, provide reproduction steps, and attach files that enable Technical Support to follow the steps.

If you do not have a MathWorks account, the login form enables you to create one. MathWorks accounts are free and without obligation. You can also use the account login form to obtain a forgotten MathWorks password. You can log out before submitting your request or remain logged in.

For an example of submitting a support request, see this video demo. For further details about requesting service requests from MATLAB, see Contact Technical Support.

Managing Files

New features and changes introduced in Version 7.11 (R2011a) are:

Renaming Files and Folders in the Current Folder Browser Now Reflected in the Editor

If a file is open in the Editor and you:

- Rename that file in the Current Folder browser, then the file name updates in the Editor
- Rename the folder containing that file in the Current Folder Browser, then MATLAB updates the file specification for all open Editor documents in the renamed folder
- Delete that file from the Current Folder browser, then MATLAB deletes the file from disk and renames the Editor document to `Untitled*`

  If more than one unnamed document is open in the Editor, then MATLAB renames the deleted document `Untitledn*`, where `n` is an integer.

However, modifying file and folder names from either of the following does not update names for open documents in the Editor:

- The operating system file manager — such as Windows Explorer
- The MATLAB Command window — using `movefile`, for instance

Options Names Changed for Locating and Opening Files and Folders Outside the MATLAB Desktop

To locate a file or folder in Windows Explorer or Apple Mac Finder, you previously used the Current Folder browser context-menu option, **Locate on Disk**. Now the option is one of the following:

- On Microsoft Windows systems—**Show in Explorer**
- On Macintosh systems—**Show in Finder**

In addition, from the Current Folder browser, you can open the current folder in Explorer or Finder. Right-click in white space, and then select **Open Current Folder in Explorer** or **Open Current Folder in Finder**.

Comparison and Merging of MAT-file Variables

When comparing MAT-files, you now can view details of differences between variables to see which fields of a structure are different and to view differences in individual elements of an array. You can merge changes between files by copying variables from one file to another. For details, see Comparing MAT-Files.

Filter Results in Folder Comparisons

You now can define filters to exclude unimportant differences when comparing folders. For example, you can exclude backup files or files created by a revision control system. Filters can save time when reviewing differences, especially when comparing many subfolders. For details, see Comparing Folders and Zip Files.

Showing Differences Only in Text Comparisons

When comparing text files, you now can specify whether to show only differences. Use the new toolbar button in the Comparison Tool to hide sections of the report that do not include any differences. It can be useful to hide unmodified lines in large text comparison reports. For details, see Comparing Text Files.

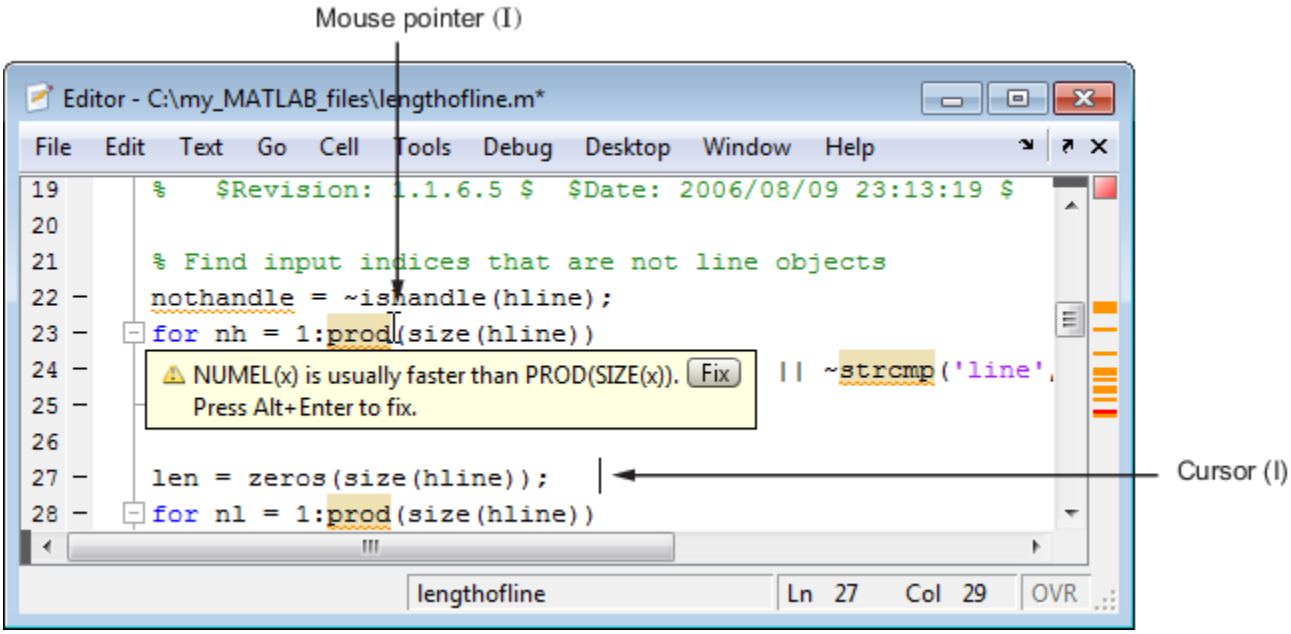
Editing and Debugging MATLAB Code

New Desktop features and changes introduced in Version 7.11 (R2011a) are:

- “Change to Tooltip and Behavior for M-Lint Messages” on page 20-4
- “Changed Default Preference for Deleting Autosaved Files” on page 20-5
- “Shared Scope Color Preferences Apply to Persistent and Global Variables” on page 20-5

Change to Tooltip and Behavior for M-Lint Messages

Previously, when you hovered the *mouse pointer* over code in the Editor that has an autofixable M-Lint warning message, a tooltip opened that included the phrase, **Press Alt + Enter to fix**, as shown in this image.



Now, this phrase is removed from the M-Lint tooltip. You can still use the keyboard shortcut, **Alt +Enter**, to apply the autofix, but your *cursor* must be within the code to which the autofix will be applied.

Compatibility Considerations

When an M-Lint tooltip is open and you want to apply the autofix for that warning, click the **Fix** button. To apply the same autofix using a keyboard shortcut, move the cursor within the code that generates the warning, and then click **Alt+Enter**.

Changed Default Preference for Deleting Autosaved Files

Previously, the **Automatically delete autosave files** preference for the Editor was disabled by default. Now, it is enabled by default. However, as usual, MATLAB retains settings you set explicitly in a previous release (unless you delete `matlab.prf`.)

For more information, see Autosaving Files and Where MATLAB Stores Preferences.

Shared Scope Color Preferences Apply to Persistent and Global Variables

Previously, the preference for **Variables with shared scope** was labeled **Nonlocal variables** and it affected the color of variables within nested functions only. Now, the Editor also applies this color preference to variables you declare as global or persistent.

This feature makes it easier to find reuses of a variable, which sometimes indicate variable scoping problems. For details, including information on how to disable or change the colors that the Editor uses for variables with shared scope, see Avoid Variable and Function Scoping Problems.

Publishing MATLAB Code

New Desktop features and changes introduced in Version 7.11 (R2011a) are:

Change to Menu Option for Including Blocks of LaTeX Code

Previously, if you wanted to include blocks of LaTeX code within your MATLAB code intended for publishing, you selected **Cell > Insert Text Markup > LaTeX Equation**. Now you select **Cell > Insert Text Markup > LaTeX Display Math**. The name of the option was changed to better distinguish it from a new publishing option that enables you to include inline LaTeX equations in your published code.

For details see LaTeX Display Math.

Menu Option to Include Inline LaTeX Math in Published MATLAB Code

The Cell menu now provides an option for including inline LaTeX equations. Position the cursor within the comment line where you want the LaTeX equation to appear, and then select **Cell > Insert Text Markup > LaTeX Inline Math**.

For details see Inline LaTeX Math Equations.

MATLAB Notebook Menu Labels

Menu labels for MATLAB Notebook are changed. To ensure your menu options are up to date, and match those presented in the documentation, run the following command:

```
notebook -setup
```

MathWorks recommends you run this command whenever you install a new version of MATLAB.

Mathematics

New Function rng

The new `rng` function controls the random number generator used by `rand`, `randi`, and `randn`. For a demonstration, see this instructional video.

`rng` is the recommended alternative to former `rand` and `randn` input syntaxes `'seed'`, `'state'` and `'twister'`. `rng` is a simpler alternative to the `RandStream` class. To use `rng`, see the documentation on Updating Your Random Number Generator Syntax.

New Function ichol

The new `ichol` function performs incomplete Cholesky factorization and is the recommended alternative to `cholinc`.

New Option for gammainc

The new options `gammainc(X,A,'scaledlower')` and `gammainc(X,A,'scaledupper')` now return scaled versions of the incomplete gamma function `gammainc`.

Performance Enhancement

- Matrix transpose
- Element-wise single precision functions
- Sparse matrix indexed assignment
- Many linear algebra functions
- Convolution for long vectors and large matrices with `conv` and `conv2`

Changes to qr

In R2010a and R2010b, the upper triangular output `R` from the full `qr` function was guaranteed to have real and nonnegative diagonal elements. In this release, the behavior reverts to that of R2009b and prior releases. That is, the diagonal of `R` may contain complex and negative elements, and will affect the unitary output `Q` correspondingly.

Compatibility Considerations

Since the QR factorization is not unique, these different results are still correct. Ensure that your code does not depend on the values of the elements of the factors `Q` and `R`.

Functionality Being Removed

Function Name	What Happens When You Use This Function	Use This Function Instead	Compatibility Considerations
`bessel`	Errors	`besselj`	Replace all instances of `bessel` with `besselj`.
`intwarning`	Errors	None	Remove all instances of `intwarning` from your code.
`luinc`	Warns	`ilu`	Replace instances of `luinc` with `ilu`.
`cholinc(X,'inf')`	Warns	None	Remove all instances of `cholinc(X,'inf')` from your code.
All other syntaxes of `cholinc` except `cholinc(X,'inf')`	Warns	`ichol`	Replace these instances of `cholinc` with `ichol`.
`RandnAlg` property of `RandStream` class	Still runs	`NormalTransform` property of `RandStream` class	Replace all existing instances of `RandnAlg` with `NormalTransform`.
`setDefaultStream` method of `RandStream` class	Still runs	`setGlobalStream` method of `RandStream` class	Replace all existing instances of `RandStream.setDefaultStream` with `RandStream.setGlobalStream`.
`getDefaultStream` method of `RandStream` class	Still runs	`getGlobalStream` method of `RandStream` class	Replace all existing instances of `RandStream.getDefaultStream` with `RandStream.getGlobalStream`.
`rand` or `randn` with the `'seed'`, `'state'` or `'twister'` inputs	Still runs	`rng`	See Updating Your Random Number Generator Syntax in the MATLAB Mathematics documentation.

Programming

Regenerate P-code Files Built Before Version 7.5

To enable the MATLAB software to take advantage of significant performance improvements planned for a future release, MathWorks recommends that you begin to regenerate any P-code files you use that were generated prior to MATLAB 7.5 (release R2007b). P-code files generated in releases earlier than R2007b will not run in this future release with the new performance features enabled.

Compatibility Considerations

Using MATLAB 7.5 or later, rebuild any P-code files that you expect to need in the future.

VideoWriter Supports Motion JPEG 2000 Files

`VideoWriter` now creates Motion JPEG 2000 (`.mj2`) files, which support logging data with these features:

- Multi-byte precision, such as 10, 12, or 16 bits
- Signed data values
- Lossless compression

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

audioplayer and audiorecorder Support Device Selection on All Platforms

`audioplayer` and `audiorecorder` now allow you to specify the input or output device on all supported platforms. In previous releases, you could only specify devices on Microsoft Windows systems.

Compatibility Considerations

- `audioplayer`, `audiorecorder`, and `audiodevinfo` now use different technology for enumerating system input and output devices. If your existing code specifies a device ID (other than the default, `-1`), check whether you need to change the ID. To determine your device IDs, call `audiodevinfo`.
- `audioplayer` does not support overlapping playback. For example, in this code, the second call to `play` returns an error:

```
chirpData = load('chirp.mat');
chirpObj = audioplayer(chirpData.y, chirpData.Fs);

gongData = load('gong.mat');
gongObj = audioplayer(gongData.y, gongData.Fs);

play(chirpObj);
play(gongObj);
```

New Class Forms the Basis for Heterogeneous Hierarchies

The `matlab.mixin.Heterogeneous` class enables you to form heterogeneous arrays containing instances of classes derived from this class. You can create heterogeneous hierarchies of both handle and value classes.

New Class Provides the Basis for Customizable Handle Object Copy Method

The `matlab.mixin.Copyable` class enables you to define handle classes that inherit a copy method whose behavior you can modify in subclasses.

MATLAB Meta-Classes Can Now Form Heterogeneous Arrays

All MATLAB meta-classes are now defined in a heterogeneous hierarchy, which enables the formation of heterogeneous arrays of meta-class objects. This capability enables functions like `findobj` and `findprop` to return heterogeneous arrays containing instances of meta-classes of different specific types. The hidden class `meta.MetaData` forms the root of the heterogeneous hierarchy.

New High-Level NetCDF Functions

MATLAB now includes several new functions that provide a high-level interface to NetCDF files. These functions let you read and write to NetCDF files, without having to use the programming paradigm required by the low-level functions in the netCDF package. Using these functions, you can create a new NetCDF file based on the schema of an existing file, convert files between NetCDF formats, and create a new NetCDF file by merging together two existing NetCDF files.

- `nccreate` — Create variable in NetCDF file
- `ncdisp` — Display contents of netCDF file
- `ncinfo` — Return information about netCDF file
- `ncread` — Read data and attributes from netCDF file
- `ncreadatt` — Read global attribute or attribute associated with variable from netCDF file.
- `ncwrite` — Write data to netCDF file
- `ncwriteatt` — Write attribute to netCDF file
- `ncwriteschema` — Add netCDF schema definitions to a NetCDF file

New High-Level HDF5 Functions

MATLAB now includes several new high-level functions for working with HDF5 files. These functions let you read and write to HDF5 files, without having to use the programming paradigm required by the low-level functions in the HDF5 package.

- `h5create` — Create HDF5 data set
- `h5disp` — Display contents of HDF5 file
- `h5info` — Return information about HDF5 file
- `h5read` — Read data from HDF5 data set.

- `h5readatt` — Read attribute from HDF5 group or data set
- `h5write` — Write to HDF5 data set
- `h5writeatt` — Write HDF5 attribute to group or data set

Compatibility Considerations

The new high-level HDF5 functions have the following compatibility considerations with the existing high-level HDF5 functions.

- `hdf5read` is not recommended. Use `h5read` instead.
- `hdf5write` is not recommended. Use `h5write` instead.
- `hdf5info` is not recommended. Use `h5info` instead.

Two New Functions Added to CDFLIB Package

The MATLAB interface to the CDF library now includes the following new functions.

- `cdflib.setFileBackward` — Sets the backward compatibility mode. If the backward mode is set to `on`, files created can be read by clients using version 2.7 of the library
- `cdflib.getFileBackward` — Return the current backward compatibility mode setting.

HDF4 Functions Grouped into Packages

The MATLAB HDF4 low-level functions and HDF EOS functions are now grouped into three new packages:

- `matlab.io.hdf4.sd` — access to more than 50 functions in the HDF library SD interface
- `matlab.io.hdfeos.gd` — access to more than 50 functions in the HDF-EOS library grid interface
- `matlab.io.hdfeos.sw` — access to more than 50 functions in the HDF-EOS library swath interface

For example, MATLAB previously included one function, `hdfsd`, that you used to call all the routines in the HDF library SD interface. Now, the new package `matlab.io.hdf4.sd` contains many individual functions that correspond to routines in the HDF SD Interface C library.

FITSREAD Function Now Supports Data Subsetting

The MATLAB `fitsread` function now supports data subsetting by using several new options: `PixelRegion`, `TableColumns`, and `TableRows`.

Unrecognized Name Warning Changed to Error

In earlier releases of MATLAB, a statement in which an unrecognized (e.g., misspelled) name immediately follows a function name and dot (`.`) results in a warning instead of an error. Here is the format this type of statement:

```
functionname.unrecognizedname
```

The following example shows such a statement case and part of the resulting warning:

```
simulink.badname

Warning: Direct access of structure fields returned by a
function call (e.g., call to simulink) is not allowed.  ...
```

In addition to being misleading, handling this case with merely a warning also allows the function (`simulink`, in this case) to execute, regardless of the fact that the name to the right of the dot is invalid.

In MATLAB version 7.12 (R2011a), this type of statement throws an error. The text of the error message is:

```
simulink.badname

??? Undefined variable "simulink" or function "simulink.badname".
```

Compatibility Considerations

Any such statements in your code that have only generated a warning message in past releases will now throw an error. Any such error that is not caught and handled appropriately will terminate the function in which it occurs. MathWorks recommends that you replace the unrecognized name to the right of the dot with the correct name.

This change also affects the output of the `lasterror` function if you attempt to access any field value (`message`, `identifier`, or `stack`) directly. Earlier versions of MATLAB generate a warning:

```
lasterror.message
Warning: Direct access of structure fields returned by a
function call (e.g., call to lasterror) is not allowed.  ...
```

This and future versions throw an error:

```
lasterror.message
??? Undefined variable "lasterror" or function "lasterror.message".
```

Regular Expressions Support Zero-Length Matching

Regular expressions in MATLAB now support successful zero-length matching. See the documentation on Empty Match mode in the description of Command Options for the `regexp` function.

Growing Arrays Is Faster

This release improves the performance of growing an array in the trailing dimension if that array has not been preallocated.

Error Checking Improved

MATLAB provides more effective error checking and returns new error messages in the following cases.

Nonstatic Method

A reference to a class method using the class name is valid only in cases where the method is static. Therefore, code of the form:

```
ClassName.ordinaryMethod
```

Where `ordinaryMethod` is not a static method of the class `ClassName`, previously returned the error: `MATLAB:class:InvalidStaticMethod`.

This code now returns the error: `MATLAB:subscripting:classHasNoPropertyOrMethod`

Nonexistent Method Name

A reference to a nonexistent method name:

```
obj.badName
```

Where `badName` is not a method defined by the class of `obj`, previously returned the error: `MATLAB:noSuchMethodOrField`.

This code now returns the error: `MATLAB:subscripting:classHasNoPropertyOrMethod`.

Compatibility Considerations

Code that checks for the specific errors previously returned must be updated to check for the new errors.

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`audioplayer(BufferLength)`	No effect (no-op)		Remove all instances of `BufferLength`.
`audioplayer(NumberOfBuffers)`	No effect (no-op)		Remove all instances of `NumberOfBuffers`.
`hdf5read`	Still runs	`h5read`	Replace with preferred function.
`hdf5write`	Still runs	`h5write`	Replace with preferred function.
`hdf5info`	Still runs	`h5info`	Replace with referred function.

Graphics and 3-D Visualization

Plot Catalog with a New Look, More Plots, and Diagnostics

The Plot Catalog GUI now offers a wider variety of plotting options. Its appearance and operation closely resemble the Plot Selector, with short descriptions of each graph type and a search box. It also categorizes graphs by type and by toolbox, and lets you designate personal Favorites (the top menu category). Access the Plot Catalog from the Plotting Tools Figure Palette, the bottom of the Plot Selector menu in the Workspace Browser, the Variable Editor, and various context menus. For a demonstration, see this instructional video.

This table lists the differences and similarities between the Plot Catalog and the Plot Selector.

	Plot Catalog	Plot Selector	Both
Window Characteristics	Opens in its own window, which persists until you close it	Opens in a pop-up window that closes after it loses focus	Show icons for plot types with descriptions and Favorites.
Help for Plot Types	Displays partial help listings from reference pages in a resizeable pane within its window	Opens a popup help window with reference information when you hover	Provide the same help content and a **More Help** hyperlink to the Help Browser.
Plot Creation and Validation	Provides a field in which you can type variable names or expressions to plot	Plots variables you select in the Workspace browser or Variable Editor	Validate input variables and display diagnoses of incorrect or insufficient inputs.
Plot Destination	Current figure or new figure	Current figure	Create figure, if none exists

Previously, the Plot Catalog did not validate input data. If you provided incorrect inputs for a plotting function, it attempted to use them, resulting in an incorrect plot or errors. Now, the Plot Catalog provides the same diagnostics as the Plot Selector. When you type workspace variable names into the Variables box on top, the tool validates variable types, sizes, and ordering. If validation fails, the Plot Selector provides a diagnostic message in its Help pane and does not let you run the plotting function.

In addition to MATLAB plots, the Plot Catalog offers the same set of choices as the Plot Selector. The choices include most types of plots from the following toolboxes (if installed):

- Control System Toolbox™
- Curve Fitting Toolbox™
- DSP System Toolbox™
- Financial Toolbox™
- Image Processing Toolbox
- Mapping Toolbox™
- Signal Processing Toolbox
- Statistics Toolbox™
- System Identification Toolbox™

For more information about the Plot Catalog, see Selecting a Graph from the Plot Catalog. For information on the Plot Selector, see “Enhanced Plot Selector Simplifies Data Display” on page 26-19.

Creating Graphical User Interfaces (GUIs)

Do not Repopulate Menus on the Mac from Inside Their Callbacks

In R2011a, figures display their menus on the Mac screen menubar instead of across the top of figure windows. Prior to R2011a, GUIs could create dynamic menus using callbacks that completely deleted, and then repopulated the contents of menus. But on a Mac, running a GUI with a menu whose callback changes all items in this manner can result in the display of a blank menu (no items). The unexpected behavior only happens when a `uimenu` callback deletes all submenus (child uimenu components) and repopulates the menu with a new set of items at the time the menu opens.

In R2011a, on the Mac platform only, `uimenu` callbacks are no longer able to replace all submenus during menu selection. Note that repopulating some (not all) menu items does not create this issue. However, it is not good programming practice to remove and insert menu items within a menu callback routine.

For more information, see “MATLAB Menus Display at the Top of the Apple Mac Screen” on page 20-2.

Compatibility Considerations

If you have a `uimenu` callback that *repopulates all of its menu items* when you open that menu, you must change that code if you want it to work on a Mac. For example, your code might be able to remove and install submenus from outside of the callback that handles the menu. The callback can also rename, disable, hide, and show submenus instead of deleting them and creating new ones. The Mac is the only platform impacted by this incompatibility. Menus on Microsoft Windows, Unix, and Linux continue to behave as they did in previous releases.

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`pagesetupdlg`	Warns	`printpreview`	`printpreview` provides all functionality and more, except for some unit choices. Update any GUIs that call `pagesetupdlg` to do page setup.

External Interfaces/API

Changes to Compiler Support

New Compiler Support

MATLAB Version 7.12 (R2011a) supports these new compilers for building MEX-files:

Linux (64- and 32-Bit) Platforms

- GNU gfortran 4.3.x

Apple Mac 64-Bit Platforms

- Apple Xcode 3.2 with gcc 4.2.x

Compiler Support To Be Phased Out

Support for the following compilers will be discontinued in a future release, at which time new versions will be supported. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Microsoft Windows 32-Bit Platforms

- Microsoft Visual Studio 2005 SP1
- Visual C++ 6.0

Windows 64-Bit Platforms

- Microsoft Visual Studio 2005 SP1

Discontinued Compiler Support

MATLAB no longer supports the following compilers:

Windows (64- and 32-Bit) Platforms

- Microsoft Visual Studio 2008 (9.0) Express
- Intel Visual Fortran Version 10.1

Linux (64- and 32-Bit) Platforms

- GNU g95 0.90

Mac 64-Bit Platforms

- Xcode 3.1 with gcc 4.0.1

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Changes to Shared Library Compiler Support

In MATLAB Version 7.12 (R2011a), you can use the `loadlibrary` command with any supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Compatibility Considerations

You must run `mex -setup` before using the `loadlibrary` command. For information about selecting a compiler, see Selecting a Compiler on Windows Platforms or Selecting a Compiler on UNIX Platforms.

New Support for Features in Microsoft .NET Framework

Support for .NET System.Enum Objects

MATLAB exposes .NET enumerations as native .NET classes:

- Support for non-`Int32` underlying types
- Support for bit-wise `bitand`, `bitnot`, `bitor`, and `bitxor` operators
- Support for invocation of `System.Enum` methods, such as the `HasFlag` method in Framework Version 4.0
- Support for comparison and binary operators `eq`, `ne`, `ge`, `gt`, `le`, and `lt` on enumeration types

See .NET Enumerations in MATLAB.

Compatibility Considerations

MATLAB displays an error when you use the `enumeration` command to return arrays of .NET enumeration objects. To read enumeration members into MATLAB arrays, see Refer to a .NET Enumeration Member.

MATLAB enumerations no longer inherit from the MATLAB `int32` class.

You cannot create arrays of .NET enumeration objects. For example, if you type:

```
a = [EnumTest.Colors.Red EnumTest.Colors.Blue]
```

MATLAB displays:

```
??? Array formation and indexing are not allowed on .NET objects.
```

To combine members of an enumeration into a MATLAB variable, see Combining Enumerations into a Single MATLAB Variable.

Support for Asynchronous .NET Delegate Callback Handling

You can use delegates to call a synchronous method asynchronously. See Calling a .NET Method Asynchronously.

R2010bSP2

Version: 7.11.2

Bug Fixes

R2010bSP1

Version: 7.11.1

Bug Fixes

R2010b

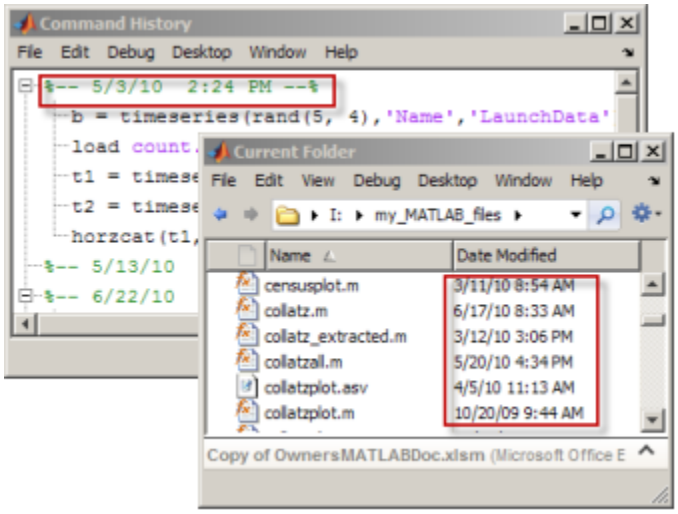
Desktop Tools and Development Environment

Desktop

New desktop features and changes introduced in Version 7.11 (R2010b) are:

Ability to Customize the Date Format

You can now customize the date format that the Current Folder browser and the Command History window use to display dates, such as the dates indicated here:



Previously, the date format in both of these tools was M/d/yy (for instance, 6/4/90 for June 4, 1990) and you could not change it. Now, both tools use the operating system short date format. For details, see Customizing the Column Display.

Keyboard Shortcuts Preferences Integrated with File Exchange

MATLAB now integrates keyboard shortcuts preferences with File Exchange. You can access File Exchange directly from the Keyboard Shortcuts Preferences panel and find:

- MATLAB keyboard shortcuts available in Version 7.9 (R2009a) and earlier releases
- Keyboard shortcuts sets created by other MATLAB users

For details, see Download Keyboard Shortcut Settings Files from File Exchange .

MATLAB Provides Enumeration Template

MATLAB provides a template for classes that define enumerations. Enumeration classes enable you to define a fixed set of names representing a particular type of value. Choose **File > New > Enumeration**. For more information, see Enumerations.

Help Browser

New Language Preference for Help Browser

If your system displays documentation in Japanese, you can use the **Language** panel in the Help Preferences dialog to instruct the Help browser to display documentation in English instead of Japanese. You can toggle between languages with the preference panel, enabling you to revert to Japanese at any time. The option changes the language used in the Help Browser and for GUI context-sensitive help, but it does *not* affect the language appearing in menus or elsewhere in products.

For example, the `help` command still provides help in Japanese after you switch the Help browser to English. The **Language** preference is available only when the system locale is Japanese and the Japanese documentation is installed. If the documentation for a product is not translated, the Help Browser displays the English documentation for it no matter how you set the preference.

Accessing Product Documentation in Japanese

MathWorks usually provides Japanese translations of product documentation about 2 months after new versions of products first ship. When you install the new version of a product, the *previous version* of translated documentation is installed in most cases.

To read the most current documentation for your release, switch the Help browser to English. After the translations of the latest documentation are available, you can download and install them in your products. At that time, you can find a link to download the latest translated documentation on the Japanese documentation start page.

Managing Files

New features and changes introduced in Version 7.11 (R2010b) are:

Ability to View Zip File Contents in Current Folder Browser

Using the Current Folder browser, you can now view the files in a zip file without having to extract them. This feature enables you to:

- Confirm the contents of a newly created zip file
- Selectively open items from a zip file
- Add or remove files from the zip file

For details, see Creating and Managing Zip File Archives or watch the Zip File Browsing video demo.

Details Panel of Current Folder Provides Preview of Graphic Files

Now, when you select a `JPEG`, `JPG`, `BMP`, `WBMP`, `PNG`, or `GIF` image in the Current Folder browser, the Details panel displays a thumbnail of the image and lists its width and height in pixels. For more information, see Viewing File Details Without Opening Files or watch the File Preview Enhancements video demo.

Current Folder Browser Indicates Whether File Is Modified in Editor

Now, if you modify a file in the Editor, but have not yet saved the changes, the Current Folder browser indicates the file state. An asterisk (*) appears next to the name of such a file in the Current Folder browser. In addition, when you select such a file in the Current Folder browser, the detail panel reflects the modified file, not the file saved on disk. This feature is useful when you are creating zip files and want to be sure that all files included in the archive are saved and up-to-date. For details see, Viewing File Details Without Opening Files or watch the File Preview Enhancements video demo.

Compare Zip Files and Folders

You can now compare any combinations of zip files, folders, and Simulink Manifests with the Comparison Tool. Right-click files or folders in the Current Folder browser and choose **Compare Selected Files/Folders**, or **Compare Against > Choose**.

For details, see Comparing Folders and Zip Files.

Enhanced Comparison Tool

The Comparison Tool now provides the following capabilities:

- Select what type of comparison to run from a list of possible comparison options, e.g., text, binary or XML comparison.
- Enhanced MAT-file comparisons now include size, data type and change summary.
- New option to ignore whitespace changes in text comparisons.

For details, see Comparing Files and Folders.

Editing and Debugging MATLAB Code

New Desktop features and changes introduced in Version 7.11 (R2010b) are:

- “Ability to Save File to Backup Without Closing That File” on page 23-4
- “Enhanced Comment Wrapping” on page 23-5
- “Variable and Function Highlighting” on page 23-5
- “Options for Setting Current Folder and Search Path Available from Editor Context Menu” on page 23-5
- “Open As Text Option” on page 23-5

Ability to Save File to Backup Without Closing That File

You now can save a file that is open in the Editor to a backup file. Select **File > Save Backup**, and then specify a backup file name. The file that is active in the Editor when you perform the save operation remains active. If changes to the active file prove problematic, you can use the backup file to return to a known state or to compare with the active file. This comparison can help you determine where errors exist.

For information on other save options, see Save Files.

Enhanced Comment Wrapping

The enhanced ability to wrap MATLAB comments includes:

- Wrapping an entire block of comments by selecting **File > Wrap Comments**.

  There is no need to select the block of comments first. For details see Wrap Comments Manually.
- Specifying where you want column counting to begin.

  For example, if you indent comments, you can specify that you want the maximum width of comments to be 75 columns from the start of a comment, rather than from the start of a line. To set **Comment formatting** preferences, select **File > Preferences > Editor/Debugger > Language**.

Variable and Function Highlighting

The Editor now presents variables that are not local variables in a teal blue color, by default. Also by default, when you click a function or local variable, the Editor highlights it and all other references to it in a sky blue color. As demonstrated in the Variable and Subfunction Highlighting video demo, this feature makes it easier to:

- Find unintentional reuses of a variable within a nested function

  For details, see Avoid Variable and Function Scoping Problems.
- Navigate through a file from function to function reference or variable to variable reference

  For details, see Navigate to a Specific Location.
- Search a file for a particular function or variable name

  For details, see Find and Replace Functions or Variables in the Current File.

To disable or change the colors that the Editor uses for variable and function highlighting, see Use Automatic Function and Variable Highlighting.

Options for Setting Current Folder and Search Path Available from Editor Context Menu

When multiple documents are open and docked in the Editor, you can right-click the document tab, and then from the context menu choose:

- **Change Current Folder to** *folder-name*
- **Add** *folder-name* **to Search Path** or **Remove** *folder-name* **from Search Path**, respectively
- **Locate on Disk**

  This option locates the document in your operating system file browser. It is not available on Linux platforms.
- **Copy Full Path to Clipboard**

Open As Text Option

You can use the **File > Open as Text** option to open a file in the Editor as a text file, even if the file type is associated with another application or tool. This feature is useful, for example, if you import a

tab-delimited data file (`.dat`) into the workspace, and then you find you want to add a data point. For details, see Open Existing Files.

Mathematics

64-Bit Integer Arithmetic

Core MATLAB arithmetic functions now support `int64` and `uint64` classes natively. Functions added are `plus` (+), `minus` (–), `uminus` (–), `times` (.*), `rdivide` (./), `ldivide` (.\), `power` (.^), `rem`, `mod`, `bitcmp`, `any`, `all`, `sum`, `diff`, `colon` (:), `sign`, `accumarray`, and `bsxfun`.

New Utility Functions: isrow, iscolumn, ismatrix

New functions `isrow`, `iscolumn`, and `ismatrix` provide basic information about inputs.

Output Option for Point Distances in DelaunayTri/nearestNeighbor Method

The two-output form of `DelaunayTri/nearestNeighbor` returns the corresponding Euclidean distances between the query points and their nearest neighbors.

Changes to convhull and delaunay Functions

The functions `convhull` and `delaunay` now support 3-D input in either multiple vector or multicolumn matrix format. In addition, the `simplify` option for `convhull` provides the option of removing vertices that do not contribute to the area or volume of the convex hull.

Performance Enhancements

- Three-Output Form of `svd`

  The three-output form of `svd` displays significantly enhanced performance.
- Sparse Column Assignment

  The assignment of elements into multiple columns of a MATLAB sparse matrix displays significantly enhanced performance.
- Trigonometric Functions

  All degree-based trigonometric functions (`sind`, `cosd`, `tand`, `cotd`, `secd`, `cscd`) and their inverses (`asind`, `acosd`, `atand`, `acotd`, `asecd`, `acscd`) display significantly enhanced performance.

Functions Being Removed

Function Name	What Happens When You Use This Function	Use This Function Instead	Compatibility Considerations
`bessel`	Warns	`besselj`	Replace all instances of `bessel` with `besselj`.

Function Name	What Happens When You Use This Function	Use This Function Instead	Compatibility Considerations
`erfcore`	Errors	`erf`, `erfc`, `erfcx`, `erfinv`, or `erfcinv`	See the function reference pages for the individual functions.
`intwarning`	Errors	None	Remove all instances of `intwarning` from your code.

optimset Errors for Optimization Toolbox Options

If you do not have an Optimization Toolbox™ license, and you set an `optimset` option for a solver that is only available in Optimization Toolbox, `optimset` errors. Previously, `optimset` would warn, not error, and ignore the option.

Compatibility Considerations

Change your code to set only those options that apply to your solver: `fminbnd`, `fminsearch`, `fzero`, or `lsqnonneg`.

atan Warning Being Removed

When you use `atan(1i)` and `atan(-1i)` you no longer get a warning.

Compatibility Considerations

Remove instances of the warning ID `MATLAB:atan:singularity` from your code.

Data Analysis

Arrays of Time Series Objects Supported

MATLAB now enables you to create arrays of `timeseries` objects. In Version 7.10 (R2010a) and before, `timeseries` objects behaved as arrays, but some array behavior was overridden.

Compatibility Considerations

Note It is likely that this change is important to you *only* if you write code that uses `timeseries` objects.

In Version 7.11 (R2010b), all of the overridden behaviors for the functions listed in the table that follows are removed. The behavior of these functions on `timeseries` objects is the built-in behavior for arrays of objects. Think of a `timeseries` object as a single object that you can concatenate into an array of objects. Do not think of each `timeseries` itself as an array.

Functions Being Modified

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`horzcat`	No error or warning	Not applicable	You can now use `horzcat` with `timeseries` objects.

`isempty (timeseries)`	No error or warning	`timeseries.Length==0`	Replace code such as:
			`isempty(ts)`
			with:
			`ts.Length == 0`

`length (timeseries)`	No error or warning	`timeseries.Length` property	Replace code such as:
			`length(ts)`
			with:
			`ts.Length`

`size (timeseries)`	No error or warning	`[timeseries.Length 1]`	Replace code such as:
			`size(ts)`
			with:
			`[ts.Length 1]`

`subsref`	No error or warning	`getsamples`	Replace code such as:
			`t3 = t1(1:3);`
			with:
			`getsamples(t1,1:3);`

`vertcat (timeseries)`	May return `Index exceeds matrix dimension` error	`append`	Replace code such as:
			`t3 = [t1;t2];`
			with
			`t3 = append(t1,t2);`

Example of timeseries Object Concatenation

Suppose you concatenate the following two `timeseries` objects:

```
t1 = timeseries(ones(5,1),0:4);
t2 = timeseries(zeros(5,1),5:9);
```

In R2010a, the following syntax created a single `timeseries` object, `t3`, with 10 samples spanning the time range 0-9. The first 5 samples derived from `t1` and the last 5 samples derived from `t2`. The size of the `timeseries`, `t3`, was `[10 1]` and its length was 10, because `t3` has 10 samples.

```
t3 = [t1;t2]
t3 = vertcat(t1,t2)
```

Now, the same syntax creates a 2x1 array comprising the two `timeseries` objects `t1` and `t2`. Its size is `[2 1]`, and its length is 2 because the array has two rows, each a single `timeseries`.

Programming

arrayfun Accepts Array of Objects

The `arrayfun` function now accepts an array of objects as an input. The output can also be a scalar object, as long as the `UniformOutput` flag is set to `false`.

Comparing Object Arrays that Contain NaNs

In previous versions of MATLAB, calling `isequalwithequalnans` to compare identical arrays of objects that contain one or more NaN (Not a Number) values incorrectly returned `false`. Similarly, calling `isequal` to compare identical arrays of objects that contain NaNs incorrectly returned `true`.

In MATLAB version 7.11, these functions return the expected values (`true` for `isequalwithequalnans`, and `false` for `isequal`) when used to compare object arrays.

Compatibility Considerations

Program code that compares object arrays that may contain NaNs should be examined to ensure correct behavior.

Functions isa and islogical Now Consistent for Objects

In earlier releases of MATLAB, the following statements returned different results for objects with a base class of `logical`:

```
isa(obj, 'logical')
islogical(obj)
```

In MATLAB version 7.11, both of these statements return `true` if the class of `obj` is derived from `logical`, and `false` otherwise.

Compatibility Considerations

If you have program code that calls either `isa` or `islogical` on objects with a base class of logical, it is advisable to verify that this new behavior does not adversely affect the execution of the program.

New Enumeration Classes

MATLAB provides support for classes that define enumerations. Enumeration classes enable you to define a fixed set of names representing a particular type of value. See Enumerations for more information.

Use this link to watch a video that introduces enumerations: Play demo

New Functionality for Writing Video Files

The new `VideoWriter` function allows you to create AVI files on all platforms. As an improvement over the `avifile` function, `VideoWriter` can create files larger than 2 GB. For more information, watch this video demo or see the `VideoWriter` reference page.

mmreader Renamed VideoReader

For consistency with the new `VideoWriter` class, the `mmreader` class is now called `VideoReader`. A new function, `VideoReader`, replaces the `mmreader` function. `VideoReader` and `mmreader` return identical `VideoReader` objects for the same input file.

Compatibility Considerations

Replace all instances of `mmreader` with `VideoReader`. The two functions use identical syntax. When requesting help or documentation for methods from the command line, or calling a static method such as `getFileFormats`, use the new `VideoReader` class name (such as `doc VideoReader/read`). You do not need to change any calls to the `read`, `get`, or `set` methods.

Previously, the R2010a Release Notes recommended replacing instances of `aviinfo` with `mmreader` and `get`, and instances of `aviread` with `mmreader` and `read`. Instead, replace `aviinfo` with `VideoReader` and `get`, and replace `aviread` with `VideoReader` and `read`.

New HDF4 Functions

The MATLAB interface to the grid functions in the HDF-EOS C library, `hdfgd`, now includes three new syntaxes. These syntaxes convert grid coordinates to longitude and latitude values.

- `ij2ll`
- `ll2ij`
- `rs2ll`

To view the help for these functions, use the `help` command at the MATLAB prompt, as in the following example:

```
help hdfgd
```

New HDF5 Low-Level Functions

The MATLAB interfaces to the HDF5 C library now include the following new functions.

- `H5A.open_by_idx`
- `H5A.open_by_name`
- `H5D.get_access_plist`
- `H5I.is_valid`
- `H5L.copy`
- `H5L.get_name_by_idx`
- `H5P.set_chunk_cache`
- `H5P.get_chunk_cache`
- `H5P.set_copy_object`
- `H5P.get_copy_object`

To view the help for these functions, use the `help` command at the MATLAB prompt, as in the following example:

```
help H5P.get_copy_object
```

New netCDF Functions

The MATLAB interface to the netCDF C library now includes the following new functions.

- `netcdf.inqFormat`
- `netcdf.inqUnlimDims`
- `netcdf.defGrp`
- `netcdf.inqNcid`
- `netcdf.inqGrps`
- `netcdf.inqVarIDs`
- `netcdf.inqDimIDs`
- `netcdf.inqGrpName`
- `netcdf.inqGrpNameFull`
- `netcdf.inqGrpParent`
- `netcdf.defVarChunking`
- `netcdf.defVarDeflate`
- `netcdf.defVarFill`
- `netcdf.defVarFletcher32`
- `netcdf.inqVarChunking`
- `netcdf.inqVarDeflate`
- `netcdf.inqVarFill`
- `netcdf.inqVarFletcher32`

To view the help for these functions, use the `help` command at the MATLAB prompt, as in the following example:

```
help netcdf.inqVarFletcher32
```

Upgrades to Scientific File Format Libraries

The following table lists upgrades to scientific file format libraries used by MATLAB.

Library	Version Upgrade
netCDF	3.6.2 to 4.0.1

New Examples in Command Line Help

The MATLAB command line help for the HDF5, netCDF, and CDF low-level functions now includes hundreds of new examples.

imread and imwrite Can Now Handle N-channel J2C JPEG 2000 Files

You can now use the `imread` and `imwrite` functions with n-channel J2C JPEG 2000 files.

J2C files are raw JPEG2000 codestreams that usually have the file extension `.j2c` or `.j2k`. J2C files don't contain color space, color palette, capture resolution, display resolution and vender specific information.

In previous releases, if you use `imread` to read a 4-channel J2C file, you receive a warning that some channels may be ignored and `imread` returns a 3-channel image. Now, `imread` returns a 4-channel image and does not issue a warning.

If you try to write a 4-channel J2C file in previous releases, `imwrite` issues an error. In this release, `imwrite` creates the 4-channel J2C file.

csvread and csvwrite Will Not Be Removed

The R2010a Release Notes originally stated that `csvread` and `csvwrite` would be removed in a future release. As of R2010b, there are no plans to remove these functions.

sprintf and fprintf Print Null Characters in Strings

In previous releases, calls to `sprintf` or `fprintf` that printed strings using the `%s` qualifier prematurely terminated strings that contained null characters. For example, this code

```
sprintf('%s', ['foo' 0 'bar'])
```

returned

```
ans =
foo
```

The same code now returns

```
ans =
foo bar
```

Compatibility Considerations

If you do not want to print the entire contents of strings that contain null characters, test the string for these characters (for example, using the `isstrprop` function).

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`fileparts` return argument 4 (file version)	Warns, returns empty fourth argument	No alternative; file versions are unsupported	Call `fileparts` with three return arguments: `[path_name file_name file_extension]`
`mmreader`	Still runs, creates a `VideoReader` object	`VideoReader`	Replace all existing instances of `mmreader` with `VideoReader`.

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`wavplay`	Still runs	`audioplayer` and `play`	Replace all existing instances of `wavplay` with `audioplayer` and `play`.
`wavrecord`	Still runs	`audiorecorder` and `record`	Replace all existing instances of `wavrecord` with `audiorecorder` and `record`.
`wk1finfo`	Warns		Remove all instances of `wk1finfo`. Get information about Excel spreadsheets with `xlsfinfo`.
`wk1read`	Warns		Remove all instances of `wk1read`. Read Excel spreadsheets with `xlsread`.
`wk1write`	Warns		Remove all instances of `wk1write`. Write to Excel spreadsheets with `xlswrite`.

MATLAB Did Not Pass struct to loadobj When Property Was Deleted

If you save an object having a property value equal to the property's default value, and then that property is removed from the class definition, MATLAB did not detect the fact that the property was missing. In this case, MATLAB did not build a `struct` to pass to `loadobj`. MATLAB now returns a `struct` in this case.

Compatibility Considerations

If you rely on the following situation *not* returning a `struct` to `loadobj`, you need to update your code:

- You saved an object using the `save` command
- This object had a property that was set to the default value defined by the class at the time of saving
- he property set to its default value at save time is subsequently removed from the class definition
- You load the property and expect the load operation to return an object (it should return a `struct` because the class definition has changed in a way the load cannot create the object).
- Your `loadobj` method is not prepared to use the returned `struct` to create an object

You must implement a `loadobj` method to recreate the object using the current class definition.

See Saving and Loading Objects for more information.

Graphics and 3-D Visualization

Print –dmfile and printdmfile Issue Deprecation Warnings

The `–dmfile` `print` command option and the `printdmfile` function now issue a deprecation warning. Both option and function will be removed in a subsequent release. The `–dmfile` option and the `printdmfile` function were used by GUIDE.

Compatibility Considerations

Remove any use of `print –dmfile` or `printdmfile` from your code.

The saveas 'mmat' option Issues a Deprecation Warning

The `saveas` function now issues a deprecation warning if you use the `'mmat'` format option. This option will be removed in a subsequent release. This option was used with GUIDE.

Compatibility Considerations

Remove any use of the `mmat` option with `saveas` from your code.

The movie Function is No Longer a Built-in Function

The `movie` function is no longer a built-in function. Therefore, you cannot call `movie` from the `builtin` function. `movie` syntax remains the same.

Compatibility Considerations

Remove any use of the `builtin` function to call the `movie` function from your code.

Creating Graphical User Interfaces (GUIs)

Functions and Function Elements Being Removed

The `uitabgroup` and `uitab` GUI components are undocumented features that provide tabbed panels to GUIs that you create programmatically. The current method of calling these functions and some of their properties will change in a future release. MathWorks has never supported their use. However, if your MATLAB code includes these functions, read the following information:

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`uitabgroup('v0',...)` `uitab('v0',...)`	Warns	`uitabgroup(...)` `uitab(...)`	Using the `'v0'` argument generates a warning, but the object is created.
`uitabgroup` property `SelectionChangeFcn`	Not recommended	`SelectionChange Callback`	Event data for the callback is still a struct with fields `OldValue` and `NewValue` but those values are now `uitab` handles, not `uitab` indexes.
`uitabgroup` property `SelectedIndex`	Not recommended	`SelectedTab`	Both properties still exist, but replace instances of `SelectedIndex` (an integer tab index) with `SelectedTab` (a handle to a `uitab` object).
`uitabgroup` property `BackgroundColor`	Not recommended	no replacement	`uitabgroup` now uses the default `uicontrol` background color.
`uitabgroup` event `SelectionChanged`	Errors	`SelectionChange` event	Event name changed to be consistent with other components. Event data is still a struct with fields `OldValue` and `NewValue` but those values are now `uitab` handles, not `uitab` indexes.

You should update any code that you maintain which uses the undocumented `uitabgroup` and `uitab` functions to conform to the new standards, as described in the uitab and uitabgroup Migration Document. Existing code for these functions is unlikely to work in R2010b going forward.

External Interfaces/API

Changes to Compiler Support

New Compiler Support

MATLAB Version 7.11 (R2010b) supports these new compilers for building MEX-files:

Microsoft Windows (64- and 32-Bit) Platforms

- Microsoft Visual Studio 2010 (10.0) Professional
- Microsoft Visual Studio 2010 (10.0) Express

Linux (64- and 32-Bit) Platforms

- GNU gcc Version 4.3.x

Compiler Support to Be Phased Out

Support for the following compilers will be discontinued in a future release, at which time new versions will be supported. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Windows (64- and 32-Bit) Platforms

- Intel Visual Fortran Version 10.1
- Microsoft Visual Studio 2005 SP1

Discontinued Compiler Support

MATLAB no longer supports the following compilers:

Windows (64- and 32-Bit) Platforms

- Intel C++ Version 9.1

Linux (64- and 32-Bit) Platforms

- GNU gcc Version 4.2.3

Apple Macintosh (32-Bit) Platforms

- Apple Xcode 3.1 (gcc / g++ Version 4.0.1)
- GNU gfortran Version 4.2.2

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

-largeArrayDims Option to MEX Will Become Default in Next Release of MATLAB

In the next release of MATLAB, the `mex` command will change to use the large-array-handling API by default. This means the `-largeArrayDims` option will become the default.

You do not need to make changes to build and run MEX-files with MATLAB Version 7.11 (R2010b).

Compatibility Considerations

Source for MEX-files built on 64-bit platforms must be updated in order to successfully build and run with the next release of MATLAB. Review your source MEX-files and `mex` build scripts. For information about migrating your MEX-files to use the large-array-handling API, see Upgrading MEX-Files to Use 64-Bit API.

MEX Function -argcheck Option Removed

Function Option Name	What Happens When you use the Option?	Compatibility Considerations
`mex -argcheck` option	Errors	Remove all existing instances of `-argcheck` in build scripts.

The `-argcheck` option was a strategy for identifying memory corruption caused by a MEX-file. When you understand the likely causes of memory corruption, you can take preventative measures in your code or use your debugger to identify the source of errors.

If a binary MEX-file causes a segmentation violation or assertion, it means the MEX-file attempted to access protected, read-only, or unallocated memory.

These types of programming errors are sometimes difficult to track down. Segmentation violations do not always occur at the same point as the logical errors that cause them. If a program writes data to an unintended section of memory, an error might not occur until the program reads and interprets the corrupted data. Consequently, a segmentation violation can occur after the MEX-file finishes executing.

One cause of memory corruption is to pass a null pointer to a function. To check for this condition, add code in your MEX-file to check for invalid arguments to MEX Library and MX Matrix Library API functions.

To troubleshoot problems of this nature, run MATLAB within a debugging environment. For more information, see Debugging C/C++ Language MEX-Files or Debugging Fortran Source MEX-Files.

MEX Function -inline Option Removed

Function Option Name	What Happens When you use the Option?	Compatibility Considerations
`mex -inline` option	Errors	Remove all existing instances of `-inline` in build scripts.

New Support for Features in Microsoft .NET Framework

MATLAB supports:

- Using .NET Delegates in MATLAB.
- Accessing Microsoft Office Applications with .NET.
- Using `System.Nullable`, described in How MATLAB Handles System.Nullable.
- Calling constructors or `NET.invokeGenericMethod` with `ref` or `out` type arguments.
- Calling methods with default arguments (`System.Reflection.Missing.Value` is supported.)

New COM Data Type Support

Support for the following COM data type has been added. See Handling COM Data in MATLAB Software for a description of supported data types.

- `VT_I8` — `signed int64`
- `VT_UI8` — `unsigned int64`
- Unsigned integer

R2010a

Desktop Tools and Development Environment

Desktop New Features Video for R2010a

For an overview of the major new features in the MATLAB Desktop Tools and Development Environment area, watch this video demo.

Startup and Shutdown

AUTOMOUNT_MAP Environment Variable No Longer Used by MATLAB

On UNIX systems, MATLAB no longer uses the `AUTOMOUNT_MAP` environment variable, the path prefix map for automounting. Previously, MATLAB set the value of the variable in `.matlab7.rc`, but you could override this setting using the `AUTOMOUNT_MAP` environment variable.

Desktop

New Desktop features and changes introduced in Version 7.10 (R2010a) are:

Enhancements for Managing Keyboard Shortcuts

Keyboard shortcuts now provide the following:

Ability to List a Set of Keyboard Shortcuts in a Text Editor

You can now list all of the keyboard shortcuts for a given set in a text editor. For details, see List All Keyboard Shortcuts in a Set.

Ability to Remove Keyboard Shortcuts Sets

You can now remove keyboard shortcut sets that you created or imported. For details, see Delete a Set of Keyboard Shortcuts.

Ability to Compare Sets of Keyboard Shortcuts

You can now compare one set of keyboard shortcuts to another. This enables you, for example, to see what is different between the current Windows default keyboard shortcuts and those that were the defaults in MATLAB Version 7.8 (R2009a) and earlier. For details, see Compare Sets of Keyboard Shortcuts.

Method for Accessing M-Lint Preferences and the M-Lint Report

The **M-Lint Preferences** is now renamed the **Code Analyzer Preference**. Therefore, to set preferences for analyzing MATLAB code, select **File > Preferences > Code Analyzer**. Similarly, the

M-Lint Report is now renamed the Code Analyzer Report. To access this report, click the Actions button  in the Current Folder browser, and then select **Reports > Code Analyzer Report**. The names of the `mlint` and `mlintrpt` functions are unchanged.

Preference for Java Heap Memory

A new preference setting allows you to adjust the amount of memory that MATLAB allocates for storing Java objects. Insufficient memory for these objects results in `OutOfMemory` errors. For more information, see Java Heap Memory Preferences.

Compatibility Considerations

Technical Solution 1-18I2C specifies a procedure for creating a text file named `java.opts` to set the Java heap size. Do not use both the new MATLAB preference and a `java.opts` file to adjust the heap size.

Help Browser

For a demonstration of previous enhancements to the Help browser, watch the Help Browser Enhancements video.

Improved Instructions and Examples for Adding Help and Demos to the Help Browser

The MATLAB documentation now includes more details about how to display your own HTML help and demos. In particular, the documentation clarifies procedures for setting up files and folders and provides three new XML file templates that you can inspect, copy, and modify.

The documentation also adds a new complete working example, called the Upslope Area Toolbox. Documentation for the toolbox includes a getting started guide, user guide, function reference pages, and release notes. The documentation set is generated entirely from MATLAB script files (also provided), using the `publish` command. The example folder includes all program files for the toolbox. However, certain routines require Image Processing Toolbox for full functionality.

You can also find the Upslope Area Toolbox on the MATLAB Central File Exchange. For more information about techniques and algorithms the toolbox uses, see the set of articles about it on this MATLAB Central blog: http://blogs.mathworks.com/steve/category/upslope-area.

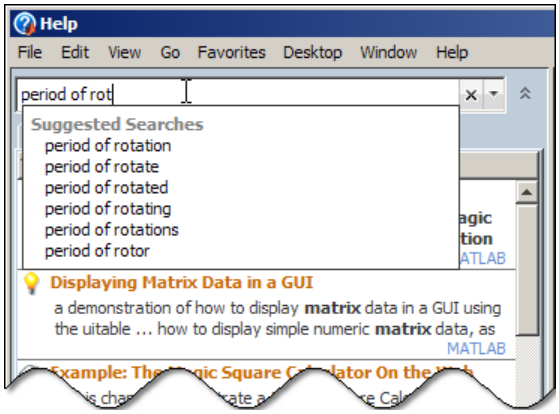
Find the updated documentation in Create Help and Demos. For details, XML templates, and examples, see these subsections:

- Add Documentation to the Help Browser
- Add Demos to the Help Browser

You can view the Upslope Toolbox documentation example in the Help browser now by clicking here, which places its folder on the search path.

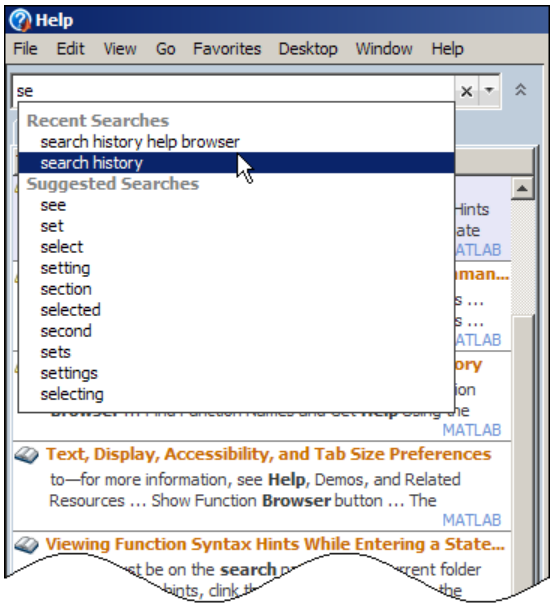
New Search Hints

The Help Browser now provides suggestions for search terms as you type in the search field. These hints save you time and can suggest words and phrases you might not have thought of but that can be useful in making a search. The hints display as a drop-down menu that changes its contents as you type. The following illustration shows the appearance of the menu.



Search History Persists Between Sessions

The Help browser can now recall searches you have made across MATLAB sessions. Your search history and search hints display as separate lists in a drop-down pane as you type in the search field. Select **Show Search History** from the pop-up menu to the right of the search field, or press the down arrow key in the search field when it is empty. The following illustration shows search history and search hints as they appear when you enter search terms.



Hide Search Results Previews

You can now choose not to see the one-line descriptions of search results called previews. To toggle all previews off, right-click in the Search Results pane of the Help Navigator and choose **Hide**

Previews. The search results listing is smaller so you can scan it quickly when you hide previews. To see previews again, right-click in the Search Results pane, and choose **Show Previews**.

docopt Function Removed

This release obsoletes the `docopt` function, which specified which system Web browser to use on UNIX platforms. This function has been removed. You can instead specify a default system browser in your preferences. For more information, see “New System Browser Preference Instead of docopt.m for MATLAB on UNIX Platforms” on page 27-2.

Compatibility Considerations

If you call `doctopt` in your startup file or some other program file, remove the call to avoid receiving an error.

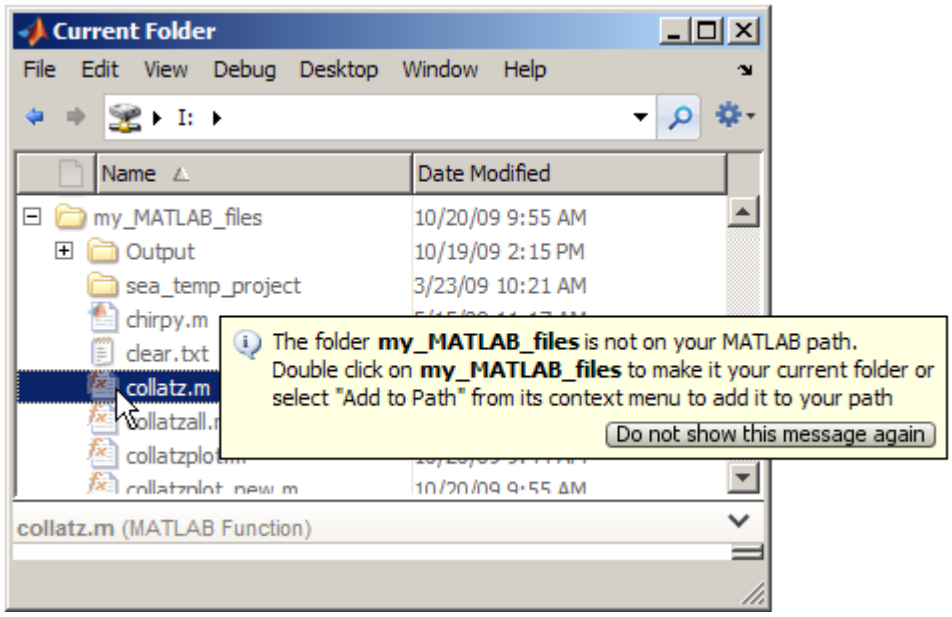
Managing Files

Create and Expand Zipped Archives from Current Folder Browser

Effective this release, the capability to create and expand, and archive `.zip` files has been added to the Current Folder browser. Select files and folders in the Current Folder browser, then right click, and choose **Create Zip File** to create an archive. The archive appears in the current folder. You can expand an archive in the same way. For details, see Creating and Managing Zip File Archives.

Visual Aids for Identifying Files Inaccessible to MATLAB

This release changes the default behavior of the Current Folder browser. The Current Folder browser now dims files that are inaccessible to MATLAB to indicate their unavailability. A tooltip provides an explanation.



You can also customize or disable this feature. For more information, see Preferences for the Current Folder Browser and Viewing Files and Folders on the Search Path.

Ability to Remove Folders and Subfolders from the Path Using the Current Folder Browser

You can now remove folders from the MATLAB path as follows:

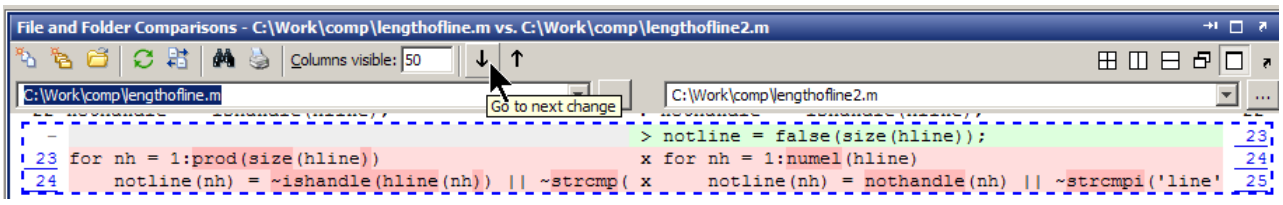
1. In the Current Folder browser, right-click the folder you want to remove from the path.
2. From the context menu, select **Remove from Path**, and then select **Selected Folders** or **Selected Folder and Subfolders**.

If the default preferences for the Current Folder **Path indication** option are set, the folders you removed now appear dimmed in the Current Folder browser. For more information, see Preferences for the Current Folder Browser and Removing Folders from the Search Path.

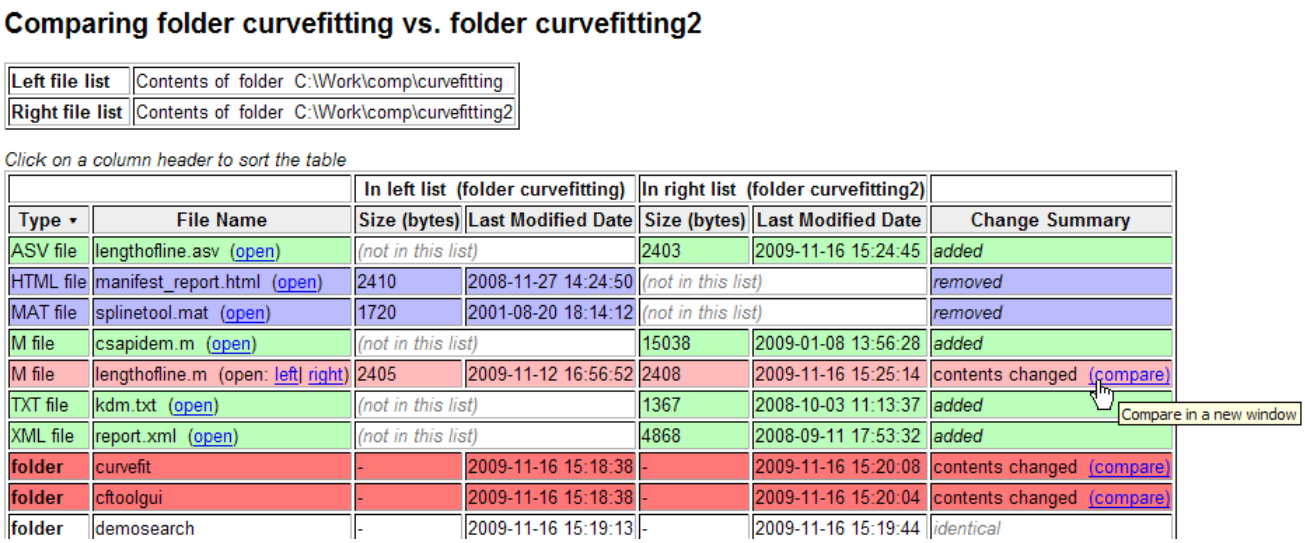
Enhancements for File and Folder Comparisons

This release enhances the File and Folder Comparisons tool to improve the usability of comparisons between text files and folders.

File comparisons now highlight changes within lines and provide new toolbar buttons for stepping through differences, as the following figure shows.



Folder comparisons now provides results that are sortable by name, type, size, timestamp, or change summary. Click column headers to sort the table. Click **compare** links to explore differences between files and folders. The following example shows results sorted by **Type**.



Comparing folder curvefitting vs. folder curvefitting2

Left file list	Contents of folder C:\Work\comp\curvefitting
Right file list	Contents of folder C:\Work\comp\curvefitting2

Click on a column header to sort the table

		In left list (folder curvefitting)		In right list (folder curvefitting2)		
Type ▾	**File Name**	**Size (bytes)**	**Last Modified Date**	**Size (bytes)**	**Last Modified Date**	**Change Summary**
ASV file	lengthofline.asv (open)	*(not in this list)*		2403	2009-11-16 15:24:45	*added*
HTML file	manifest_report.html (open)	2410	2008-11-27 14:24:50	*(not in this list)*		*removed*
MAT file	splinetool.mat (open)	1720	2001-08-20 18:14:12	*(not in this list)*		*removed*
M file	csapidem.m (open)	*(not in this list)*		15038	2009-01-08 13:56:28	*added*
M file	lengthofline.m (open: left right)	2405	2009-11-12 16:56:52	2408	2009-11-16 15:25:14	contents changed (compare)
TXT file	kdm.txt (open)	*(not in this list)*		1367	2008-10-03 11:13:37	*added*
XML file	report.xml (open)	*(not in this list)*		4868	2008-09-11 17:53:32	*added*
folder	curvefit	-	2009-11-16 15:18:38	-	2009-11-16 15:20:08	contents changed (compare)
folder	cftoolgui	-	2009-11-16 15:18:38	-	2009-11-16 15:20:04	contents changed (compare)
folder	demosearch	-	2009-11-16 15:19:13	-	2009-11-16 15:19:44	*identical*

When you compare subfolders with many files, now the tool continues analysis in the background, reporting progress. You can now skip items or cancel as you review this analysis.

For more information, see Comparing Files and Folders.

Editing and Debugging MATLAB Code

New features and changes introduced in Version 7.10 (R2010a) are:

Tab Completion for Local Variables and Functions

The MATLAB Editor now supports tab completion for local variable and subfunction names within MATLAB program files. For more information, see Complete Names in the Editor Using the Tab Key.

Toolbar Buttons for Stepping Through Code Cells Without Evaluating Code

This release provides two New Cell Mode toolbar buttons, Next Cell and Previous Cell. You can use these buttons to step through cells in the Editor without evaluating the code within the cells. By default, these buttons do not appear on the Cell Mode toolbar.

For more information, see Navigate Among Code Cells in a File.

Mathematics

New Multithreading Capability

MATLAB's new multithreading capability now includes:

- `fft` for long vectors
- The two-input form of `conv2`
- Integer conversion and arithmetic

Performance Improvements

MATLAB features significant performance improvements for:

- Sparse matrix indexed assignment and deletion
- `bvp4c` and `bvp5c` for sparse problems
- `sortrows`
- `mrdivide` (/)
- `convn`
- `histc`

Changes To qr

The factorization routine in `qr` produces an upper triangular matrix, `R`. Now this matrix always contains real and nonnegative diagonal elements. In previous releases, the diagonal of `R` could contain complex and negative elements.

Compatibility Considerations

`R` now contains only real, nonnegative diagonal elements. The `QR` factorization is not unique, so the answer is still correct.

Change in Indexing for Sparse Matrix Input

Now subscripted reference into a sparse matrix always returns a sparse matrix. In previous versions of MATLAB, using a double scalar to index into a sparse matrix resulted in full scalar output.

Compatibility Considerations

Scalars extricated from sparse matrices are no longer full. If you previously used the output with a function that does not support sparse input, you now need to update your code. For a list of the functions that do not support sparse matrices, see Functions That Do Not Support Sparse Matrices in the MATLAB Mathematics documentation.

Improved Error Checking for Sparse Functions

Functions that erroneously accepted N-D arrays and returned reshaped sparse 2-D outputs no longer accept full N-D inputs and now return an error. The functions affected are:

- `spones`, `spfun`, `sprand`, `sprandn`, and `sprandsym`
- `cat`, `horzcat`, and `vertcat`

Also, binary functions that formerly accepted both full N-D inputs along with sparse 2-D inputs and warned about the reshape, now return an error. For example, `ones(2,2,2).*sparse(3)` formerly returned a sparse 2-by-4 matrix along with a warning about the reshape. This code now returns an error.

Computational Geometry Functions Being Changed

The computational geometry functions `delaunay`, `convhull`, `griddata`, `voronoi`, `delaunay3`, `griddata3` no longer use the `options` arguments.

Compatibility Considerations

Use of the `options` arguments to `delaunay`, `convhull`, `griddata`, and `voronoi` now throws a warning.

Computational Geometry Functions Being Removed

The functions `griddata3`, `delaunay3`, `dsearch`, `tsearch` will be removed in a future release. Use of these functions now throws a warning.

Function Name	Use This Method Instead
`griddata3`	`TriScatteredInterp`
`dsearch`	`DelaunayTri/nearestNeighbor`
`tsearch`	`DelaunayTri/pointLocation`
`delaunay3`	`DelaunayTri`

Compatibility Considerations

Update your code to use the `DelaunayTri` and `TriScatteredInterp` computational geometry classes .

lsqnonneg No Longer Uses Optional Starting Point Input

The `lsqnonneg` solver no longer uses the optional input `x0` as a starting point.

Compatibility Considerations

If you give a starting point `x0`, MATLAB issues a warning. Also, `lsqnonneg` ignores `x0`, and instead uses a starting point of a vector of zeros.

Function erfcore Removed

Support for the `erfcore` function has been removed. Use of `erfcore` now results in the following error message:

```
ERFCORE will be removed in a future release.
Use ERF, ERFC, ERFCX, ERFINV, or ERFCINV instead.
```

Compatibility Considerations

Replace `erfcore` with `erf`, `erfc`, `erfcx`, `erfinv`, or `erfcinv`. See the function reference pages for the individual error functions.

Integer Warning Messages Removed

These warning messages for integer math and conversion have been removed:

- `MATLAB:intConvertNaN`
- `MATLAB:intConvertNonIntVal`
- `MATLAB:intConvertOverflow`
- `MATLAB:intMathOverflow`

Compatibility Considerations

The warning messages for integer math and conversion are no longer available. Remove these warning IDs from your code.

Function intwarning Being Removed

The `intwarning` function will be removed in a future release. Use of `intwarning` now throws a warning:

```
Warning: All four integer warnings are removed.
INTWARNING will be removed in a future release.
```

Compatibility Considerations

The warnings previously thrown when you used `intwarning` on are now removed. Remove all instances of `intwarning` from your code.

atan Warning Being Removed

The warning thrown when you use `atan(i)` and `atan(-i)` will be removed. Currently, the warning message is:

```
Warning: Singularity in ATAN. This warning will be
removed in a future release.Consider using DBSTOP
IF NANINF when debugging.
```

Compatibility Considerations

Remove instances of the warning ID `MATLAB:atan:singularity` from your code.

nextpow2 Returns Output the Same Size As Input

In previous releases, `nextpow2` with a vector input `v` returned `nextpow2(length(v))`. Now `nextpow2(v)` returns a vector the same size as the input.

Compatibility Considerations

If you require the old behavior, replace instances of `nextpow2` with `nextpow2(length(v))`.

Math Libraries Not Available to Build MEX-Files with Compaq Visual Fortran

MATLAB no longer provides the `libdflapack.dll` and `libdfblas.dll` math libraries for building Fortran MEX-files with a Compaq® Visual Fortran compiler.

Compatibility Considerations

If you distribute MEX-files that call BLAS or LAPACK functions built on a Compaq Visual Fortran compiler, you must also distribute the `libdflapack.dll` and `libdfblas.dll` library files.

If MATLAB displays errors when loading a MEX-file and the Dependency Walker indicates missing `libdflapack.dll` and/or `libdfblas.dll` libraries, contact the MEX-file vendor for copies of the libraries. The third-party product, Dependency Walker, is available from the Web site `http://www.dependencywalker.com`.

Data Analysis

Operations on Timeseries Objects Sometimes Warn About the isTimeFirst Property

In a future release, the `timeseries` Boolean property `isTimeFirst` will behave differently. If the value is `true`, this property indicates that the time vector is the *first* dimension of a time series. If the value is `false`, the time vector is the *last* dimension of a time series. In this release, you receive a warning that certain settings of `isTimeFirst` will not be valid in a future release. The warning indicates that for 3-D or N-D data, time must be the *last* dimension, while 2-D time series data can have time as the first dimension.

Compatibility Considerations

You can receive a warning when:

- Constructing a time series object with three or more dimensions without specifying a time vector. In a future release, MATLAB will calculate a value of the `isTimeFirst` property that can differ from the value it currently calculates.
- Changing the `data` of a time series to data of a different dimensionality. In a future release MATLAB will calculate a value of the `isTimeFirst` property that differs from the value it currently calculates.
- Directly setting the `isTimeFirst` property for a time series to a value that will be invalid in a future release.

The warning does not affect how you can use time series data in this release. It is intended to inform you that the behavior of `isTimeFirst` will change in a future release. If you receive a warning about `isTimeFirst`, you can use the `timeseries` method `migrateistimefirst` to correct the problem. For information type

```
help timeseries/migrateistimefirst
```

in the Command Window.

Time Series Time Vectors Can Now Contain Duplicate Sample Times

Starting with this release, a vector containing sample times can have duplicate times in contiguous positions. Previously, time vectors needed to increase monotonically and duplicated time values generated errors. This condition is now relaxed, such that time vectors must be *nondecreasing*. Interpolation of time series can produce duplicated times if the input to the interpolation method contained duplicate time samples. For more information, see the `timeseries` reference documentation.

Programming

Subscripting Into Function Return Values

If you have a function, such as the following, that returns a struct array:

```
function structOut = getStruct
structOut = struct('fieldA', 5, 'fieldB', 10);
```

it is no longer valid to access fields of the structure by directly dot indexing the function's return value, as shown here:

```
getStruct.fieldA
```

Instead, you should first assign the returned structure to a variable, and then dot index the variable:

```
s = getStruct;
s.fieldA
ans =
     5
```

This type of field access has never been allowed within function code, and this change causes scripts and command line statements to be in agreement with function rules.

Compatibility Considerations

If you have scripts that employ this type of dot indexing, they will now throw an error. Replace this unsupported style of dot indexing with the style shown above.

New Constructor for Map Containers

The following command constructs an empty `containers.Map` object that uses a key type of `kType`, and a value type of `vType`:

```
M = containers.Map('KeyType', kType, 'ValueType', vType)
```

See the `containers.Map` function reference page for more information.

Function Handle Access to Private and Protected Methods

When creating a function handle inside a method of a class, the function is resolved using the permissions of that method. When MATLAB invokes the function handle, it does so using the permissions of the class. This gives MATLAB the same access as the locations where the function handle was created, including access to private and protected methods accessible to that class.

See Obtaining Permissions from Class Methods for more information.

Listing Video File Formats Supported by mmreader

The new `mmreader.getFileFormats` method returns a list of the formats that `mmreader` supports. The list of formats is static for Windows and UNIX systems, but dynamic on Macintosh systems. For more information, see the `getFileFormats` reference page.

unzip Preserves Write Attribute of Files

In previous releases, for files archived using WinZip® software, the `unzip` function set the file write attribute of extracted files to read only. Starting with R2010a, `unzip` preserves the original attribute.

New Package Provides Access to low-level CDF API Routines

MATLAB has included high-level routines for accessing Common Data Format (CDF) files: `cdfread`, `cdfwrite`, and `cdfinfo`. However, the CDF API is so large, these functions cannot satisfy every need. To solve this, MATLAB now includes a new package, called `CDFlib`, that provides access to dozens of the functions in the CDF API. Using these low-level functions, you can create CDF files and populate them with variables and attributes using the CDF library, version 3.3.0.

Upgrades to Scientific File Format Libraries

The following table lists upgrades to several scientific file format libraries used by MATLAB.

Library	Version Upgrade
CDF	3.2.1 to 3.3.0
HDF5	1.8.1 to 1.8.3
HDF4	4.2.r1 to 4.2.r4
HDF-EOS2	2.8 to 2.16
PNG	1.2.8 to 1.2.39

Tiff Class Enhancements

The Tiff class includes the following enhancements.

Tiff Class Now Excludes Reading OJPEG Format Image Data

You can no longer read YCbCr OJPEG ("old-style" JPEG compression) TIFF images. The Tiff object was not intended to work with OJPEG data but some TIFF files contain data in this format. Reading this data could cause LibTIFF to terminate.

Compatibility Considerations

To read OJPEG image data, use the `imread` function. Note that the `imread` function transforms the image data into the RGB colorspace.

Additional Reading and Writing Capabilities

The Tiff class includes the following additional capabilities:

- Writing image data that use 16-bit colormaps (palettes)
- Reading and writing `LogL` and `LogLUV` High Dynamic Range (HDR) images. To write a `LogL` or `LogLUV` image, you must use the new `SGILogDataFmt` property.

MATLAB Adds Support for Creating JPEG 2000 Files

You can now use the `imwrite` function to create a JPEG 2000 file and write data to the file. The `imwrite` function supports format-specific parameters that let you specify the mode, compression ratio, and other characteristics.

Sealed No Longer Listed as meta.property Class Property

While class properties do not have a `Sealed` attribute, the `meta.property` class listed `Sealed` as a property. `Sealed` is no longer listed as a `meta.property` class property.

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations
`aviread`	Warns	`mmreader` and `read`	Replace all existing instances of `aviread` with `mmreader` and `read`.
`aviinfo`	Warns	`mmreader` and `get`	Replace all existing instances of `aviinfo` with `mmreader` and `get`.
`exifread`	Warns	`imfinfo`	Replace all existing instances of `exifread` with `imfinfo`
`fileparts`	Still runs, *`versn`* output is always empty (`''`)		Remove all references to a fourth output (*`versn`*) from `fileparts`.
`intwarning`	Warns	`warning`	If you use `intwarning`, you can implement the same warnings using the `warning` function.
`isstr`	Still runs	`ischar`	Replace all existing instances of `isstr` with `ischar`.
`mmreader.isPlatformSupported`	Warns, always returns `true`		For a platform-specific list of supported video file formats, use `getFileFormats`.
`setstr`	Still runs	`char`	Replace all existing instances of `setstr` with `char`.
`str2mat`	Still runs	`char`	Replace all existing instances of `str2mat` with `char`.

Function or Function Element Name	What Happens When You Use the Function or Element?	Use This Instead	Compatibility Considerations

`strread`	Still runs	`textscan`	Replace all existing instances of `strread` with `textscan`. For example, replace `[a,b,c] = strread(...)` with
			`C = textscan(...)`
			`[a,b,c] = deal(C{:})`
			Unlike `strread`, the `textscan` function converts numeric values to the specified data type, allowing preservation of integer types.

`strvcat`	Still runs	`char`	Replace all existing instances of `strvcat` with `char`. Unlike `strvcat`, the `char` function does *not* ignore empty strings.
`textread`	Still runs	`textscan`	Replace all existing instances of `textread` with `textscan`, similar to `strread`. Open and close files with `fopen` and `fclose`.
`wk1finfo`	Still runs		Remove all instances of `wk1finfo`. Get information about Excel spreadsheets with `xlsfinfo`.
`wk1read`	Still runs		Remove all instances of `wk1read`. Read Excel spreadsheets with `xlsread`.
`wk1write`	Still runs		Remove all instances of `wk1write`. Write to Excel spreadsheets with `xlswrite`.

Graphics and 3-D Visualization

Plot Selector Supports Additional Toolboxes

The Plot Selector, which was upgraded in Version 7.9 (R2009b), now also generates plots for display functions in the following toolboxes:

- Curve Fitting Toolbox
- DSP System Toolbox
- Image Processing Toolbox
- Signal Processing Toolbox

The Plot Selector continues to support display functions for Control System Toolbox, Financial Toolbox, and Statistics and Machine Learning Toolbox™. You must have installed a toolbox to display data using its plotting functions. For more information about the Plot Selector, see “Enhanced Plot Selector Simplifies Data Display” on page 26-19.

External Interfaces/API

Changes to Compiler Support

New Compiler Support

MATLAB Version 7.10 (R2010a) supports these new compilers for building MEX-files:

Windows (32-Bit) Platforms

- Intel C++ Version 11.1
- Intel Visual Fortran Compiler Professional Edition for Windows Version 11.1, installed with Microsoft Visual Studio 2008 Shell. This product is bundled.
- Intel Visual Fortran Compiler Professional Edition for Windows Version 11.1 and Microsoft Visual Studio 2008 SP1 Professional Edition. Separate products.
- Open Watcom Version 1.8

Windows (64-Bit) Platforms

- Intel C++ Version 11.1
- Intel Visual Fortran Compiler Professional Edition for Windows Version 11.1, installed with Microsoft Visual Studio 2008 Shell. This product is bundled.
- Intel Visual Fortran Compiler Professional Edition for Windows Version 11.1 and Microsoft Visual Studio 2008 SP1 Professional Edition. Separate products.

Compiler Support to Be Phased Out

Support for the following compilers will be discontinued in a future release, at which time new versions will be supported. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Windows (32-Bit) Platforms

- Intel Visual Fortran Version 10.1
- Intel C++ Version 9.1

Windows (64-Bit) Platforms

- Intel Visual Fortran Version 10.1
- Intel C++ Version 9.1

Discontinued Compiler Support

MATLAB no longer supports the following compilers:

Windows (32-Bit) Platforms

- Open Watcom Version 1.7
- Microsoft Visual Studio .NET 2003 Version 7.1

Solaris SPARC (64-Bit) Platforms

- Sun™ Studio 11 cc / CC Version 5.8

- Sun Studio 11 f90 Version 8.2

Compatibility Considerations

To ensure continued support for building your MEX-files, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Changes to Libraries on Linux with Debian Systems

If you run MATLAB Version 7.10 (R2010a) on a Linux with Debian® system, you must use Debian 5, as indicated in System Requirements - Release 2010a.

MEX-files created with MATLAB Version 7.2 (R2006a) or earlier depend on the `libstdc++.so.5` library. Debian 5 does not include this library. MATLAB running on Linux platforms with Debian 5 cannot load these pre-R2006a MEX-files.

Compatibility Considerations

You cannot run MEX-files created with MATLAB R2006a or earlier on Linux with Debian 5. Recompile these files on a system with Debian 5.

Math Libraries Not Available to Build MEX-Files with Compaq Visual Fortran

MATLAB no longer provides the `libdflapack.dll` and `libdfblas.dll` math libraries for building Fortran MEX-files with a Compaq Visual Fortran compiler.

Compatibility Considerations

If you distribute MEX-files that call BLAS or LAPACK functions built on a Compaq Visual Fortran compiler, you must also distribute the `libdflapack.dll` and `libdfblas.dll` library files.

If MATLAB displays errors when loading a MEX-file and the Dependency Walker indicates missing `libdflapack.dll` and/or `libdfblas.dll` libraries, contact the MEX-file vendor for copies of the libraries. The third-party product, Dependency Walker, is available from the Web site `http://www.dependencywalker.com`.

Cannot Create MEX-Files with DLL File Extension

On Windows 32-bit platforms, support for MEX-files with a `.dll` file extension is being phased out. Use the `.mexw32` extension instead.

MATLAB Version 7.10 will continue to execute `.dll` MEX-files, but future versions of MATLAB will not support the `.dll` extension. You can no longer use the `mex` function `-output` switch to create MEX-files with a `.dll` extension. If you enter a command such as:

```
mex myfile.c -output newfile.dll
```

MATLAB creates the MEX-file `newfile.mexw32` and displays a warning message.

For more information about MEX-files with `.dll` extensions, see Running MEX-Files with .DLL File Extensions on Windows 32-bit Platforms.

Compatibility Considerations

Recompile MEX-files with `.dll` file extensions. Update MATLAB scripts or `makefiles` that explicitly specify `.dll` extensions for compiled MEX-files.

If you use MEX-files with a `.dll` extension from a third-party source, contact that vendor to get a recompiled version, referring to these release notes.

-largeArrayDims Option to MEX Will Become Default in Next Release of MATLAB

In the next release of MATLAB, the default `mex` command will change to use the large-array-handling API. This means the `-largeArrayDims` option will become the default.

You do not need to make changes to build and run MEX-files with MATLAB Version 7.10 (R2010a).

Compatibility Considerations

Source for MEX-files built on 64-bit platforms must be updated in order to successfully build and run with the next release of MATLAB. Review your source MEX-files and `mex` build scripts. For information about migrating your MEX-files to use the large-array-handling API, Upgrading MEX-Files to Use 64-Bit API.

R2009bSP1

Version: 7.9.1

Bug Fixes

New features and changes introduced in this version are as follows:

The version number of the MATLAB Compiler Runtime (MCR) in release R2009bSP1 is different from the MCR version number in release R2009b. For details about MCR version numbers and corresponding MATLAB releases, see `https://www.mathworks.com/support/solutions/en/data/1-4GSNCF/?solution=1-4GSNCF`.

R2009b

Desktop Tools and Development Environment

Startup and Shutdown

Changes to -nodisplay and -noFigureWindows Startup Options

When you start MATLAB using `-nodisplay` (UNIX) or `-noFigureWindows` (Microsoft Windows) startup options, running a built-in GUI (predefined dialog boxes) generates this warning:

```
This functionality is no longer supported under
the -nodisplay and -noFigureWindows startup options.
```

If the GUI is modal, MATLAB execution suspends, and you need to type **Ctrl+C** for the Command Window prompt to reappear.

This change affects predefined dialog boxes, such as `dialog`, `msgbox`, `printpreview`, `uigetfile`, `uisetcolor`, `waitfor`, `waitbar`, and more. For a list of predefined dialog boxes, see Predefined Dialog Boxes. For more information on startup options, see the `matlab (UNIX)` and `matlab (Windows)` reference pages, respectively.

In a future release, instead of a warning, MATLAB will generate an error when you use such a dialog box under the `-nodisplay` or `-noFigureWindows` options.

Compatibility Considerations

To avoid generating the warning, start MATLAB without the `-nodisplay` or `-noFigureWindows` startup options. To avoid warnings while continuing to use these startup options, remove the code that is now producing the warnings.

Changes to Memory Manager Startup Options

Version 7.9 removes support for the `-memmgr` and `-check_malloc` command line arguments and the `MATLAB_MEM_MGR` environment variable.

Compatibility Considerations

MATLAB ignores any memory manager options.

Desktop

New features and changes introduced in Version 7.9 (R2009b) are:

Ability to Customize Keyboard Shortcuts

MATLAB supports customizing keyboard shortcuts for desktop tools. Press combinations of keyboard keys to perform a desktop action, instead of using the mouse to select items from menus. For example, press **Ctrl+N** to open a blank file in the Editor.

The ability to customize keyboard shortcuts enables you to:

- Modify keyboard shortcuts across desktop tools on an action-by-action basis.
- Modify keyboard shortcuts to match other applications you use or to match your personal preferences.
- Share your keyboard shortcuts with others, or use shortcuts created by someone else.
- Create keyboard shortcuts for some actions that currently have no shortcut.
- Use your preferred keyboard shortcuts when you use MATLAB on a different system than you typically use.

To customize keyboard shortcuts, choose **File > Preferences > Keyboard > Shortcuts**.

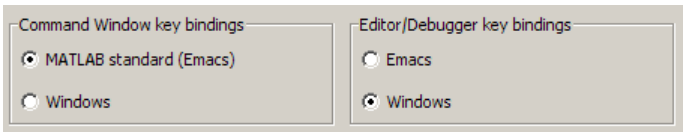
MATLAB does not support customizing keyboard shortcuts for Figure Windows, dialog boxes, or toolboxes. For details, see Customize Keyboard Shortcuts. For a video demo, see Customizable Keyboard Shortcuts.

Compatibility Considerations

- The method for specifying keyboard shortcuts differs from previous versions.

  In previous versions, you chose **File > Preferences > Keyboard**, and then chose a set of key bindings. MATLAB no longer limits your choices to either platform-specific settings or Emacs settings.
- Some default keyboard shortcuts differ from the previous defaults.

  Defaults changed in Version 7.9 (R2009b) to be more consistent across the MATLAB desktop. Previously, you could specify keyboard shortcut preferences for the Command Window and Editor/Debugger only. For instance, on Windows, default keyboard shortcut preferences appeared as follows:

  To restore keyboard shortcuts that were the default before Version 7.9 (R2009b):

  1. Choose **File > Preferences > Keyboard > Shortcuts**.
  2. From the **Active settings** drop-down menu, choose **R2009a Windows Default Set**, **R2009a Macintosh Default Set**, or **R2009a UNIX Default Set**, depending on the platform on which MATLAB is installed.

Ability to Set Fonts Preferences for Extended M-Lint Messages and Function Browser

MATLAB now supports setting the font size and type for extended M-Lint messages and the function browser. In previous releases, changes to **HTML Proportional Text** affected only the display in the MATLAB Web browser, the Profiler, and Help topics. To set font preferences:

1. Choose **File > Preferences > Fonts > Custom**.
2. Under **Desktop tools**, select **HTML Proportional Text**, and then specify the font size and type you want to use.

For more information, see Fonts.

Save Files from MATLAB Web Browser

To save a file being displayed in the MATLAB Web Browser, select **File** > **Save As**.

Help Browser

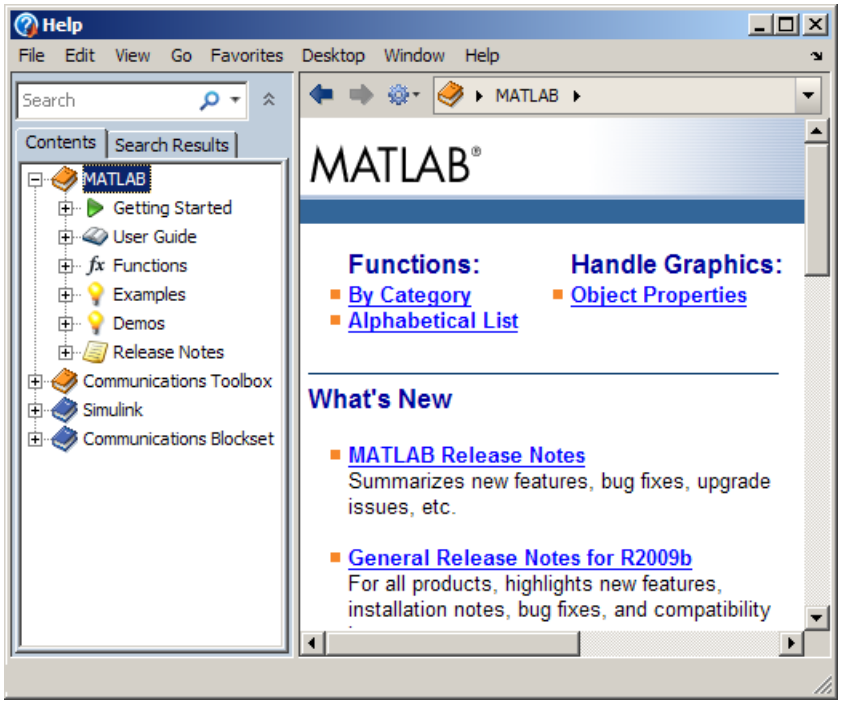
For a demonstration of enhancements to the Help browser, watch the Help Browser Enhancements video.

Improved Contents Listing

When you expand a product in the **Contents** pane, you now can view the following:

- Function and block names with brief descriptions — Expand **Functions** or **Blocks**, and then expand a category to see the list and descriptions. Select a function or block to view its reference page.

  If you provide your own HTML help files for use in the Help browser, you now can include **Functions** and **Blocks** entries for your toolbox.
- Demos — Expand **Demos**, and then select a demo from the list to view or run it.



Enhanced Presentation of Search Results

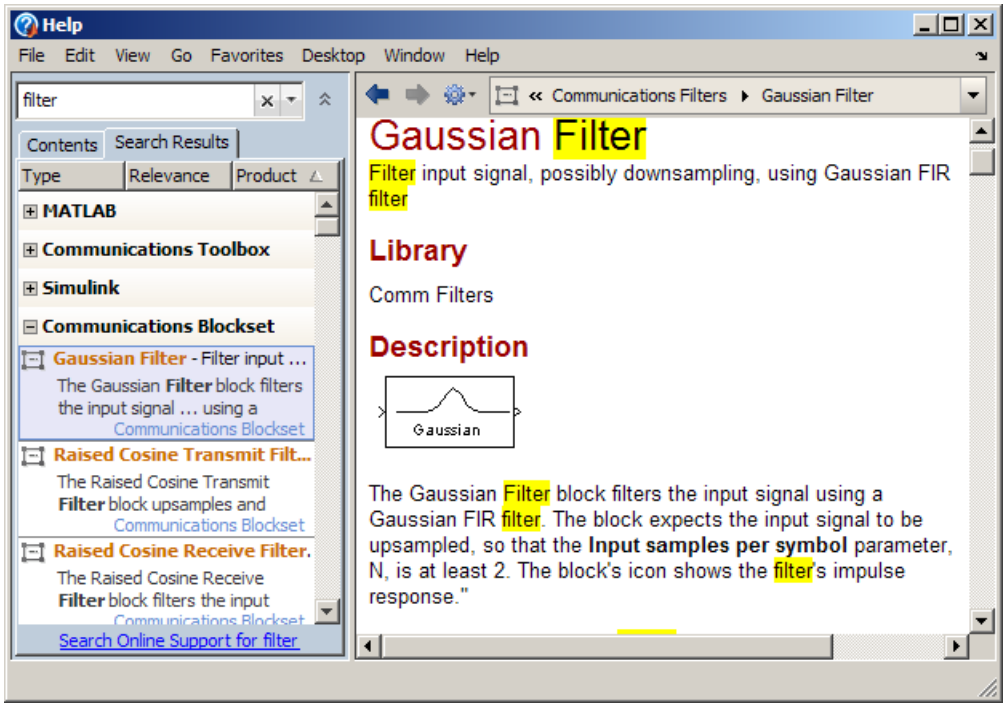
Each search result includes:

- A preview of where the search words were found within the page.
- An icon representing the type of document, such as a reference page or demo.

Use new sorting and grouping features to arrange results:

- Sort results by **Relevance**, in addition to sorting by **Type** and **Product**, which were available in previous versions.
- After sorting by **Type** or **Product**, you now can collapse and expand results for each type or product. To expand or collapse all groups, right-click in the results pane and select the option you want from the context menu.

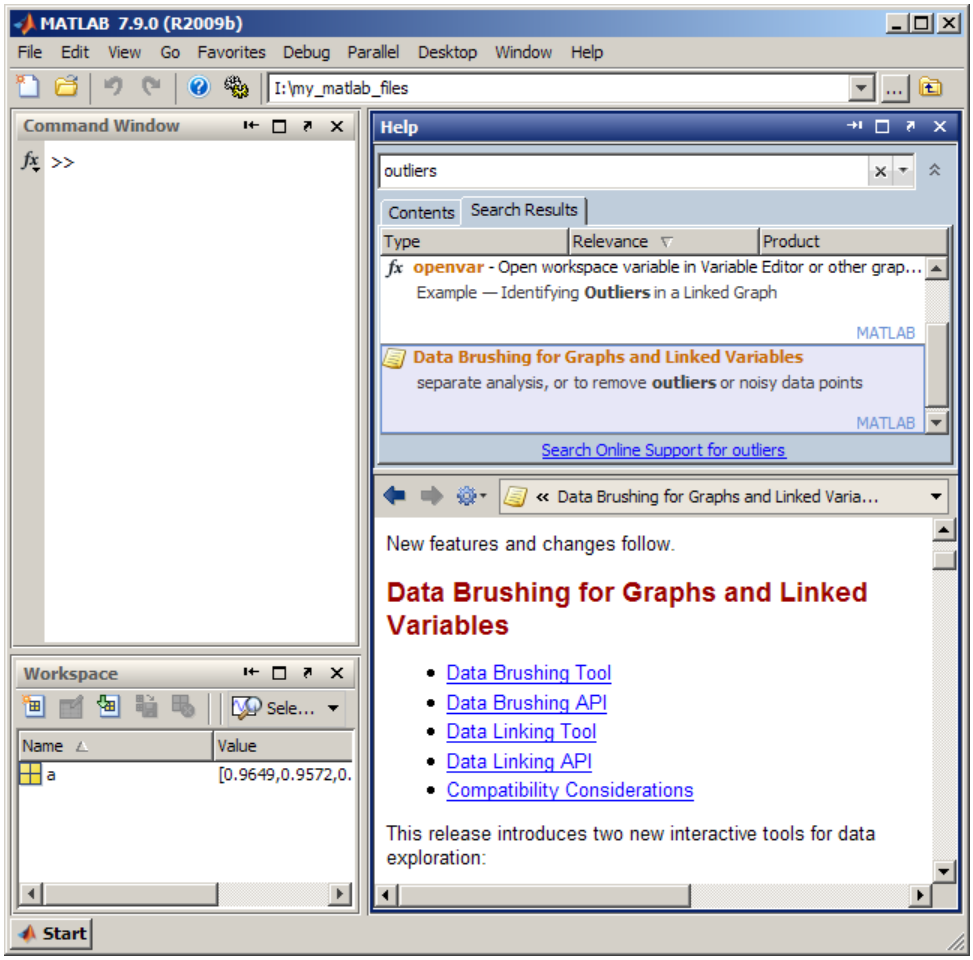
The following example of **Search Results** in the Help browser illustrates grouping, previews, and the block reference page icon.



Viewing Pages

- To see where the current page is located within the documentation, use the navigation bar at the top of the display pane. To go to another topic in the documentation, select an entry from the navigation bar.

- Use the new Actions button  on the display pane toolbar to access features such as Refresh, which clears the search results highlighting for the current page.
- Get links to reference pages for overloaded functions. When you run `doc foo`, if `foo` is an overloaded function, a message appears at the top of the display pane that provides links to the other `foo` reference pages.
- To use the Help browser alongside other tools, dock it in the desktop. When docked in a narrow area of the desktop, the pane for **Contents** and **Search Results** moves above the display pane.



Compatibility Considerations

The **Index** and **Demos** tabs are no longer in the Help browser:

- To find terms that were in the **Index**, use the search feature instead.
- To access demos for a product, go to the **Contents** pane and expand the **Demos** entry for the product.

The Actions button on the toolbar provides access to features for the displayed page. Previously, some of these features were available on individual toolbar buttons. To add individual toolbar buttons, right-click the toolbar and select **Customize**.

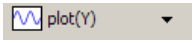
If you provide your own files for use in the Help browser:

- Remove the `helpindex.xml` entry from the `info.xml` file for your toolbox. MATLAB no longer supports the **Index**.
- Demos you add now appear in the **Contents** pane under `Other Demos`, which is after the entries for all MathWorks products.

For more information about the new Help browser features, see Help and Product Information.

Workspace and Variable Editor

Improved Workspace Plotting Tool

The Plot Selector button on 

the Workspace Browser and Variable Editor toolbars has a new look and added capabilities, options, and help. The tool now displays larger icons, includes many more graphing functions, summarizes each function, and displays pop-up windows with syntax descriptions (function hints on page 28-5). You can customize the tool by rearranging and categorizing functions, and by creating a list of “favorites”.

For further details, see this video demo and the release note “Enhanced Plot Selector Simplifies Data Display” on page 26-19.

Managing Files

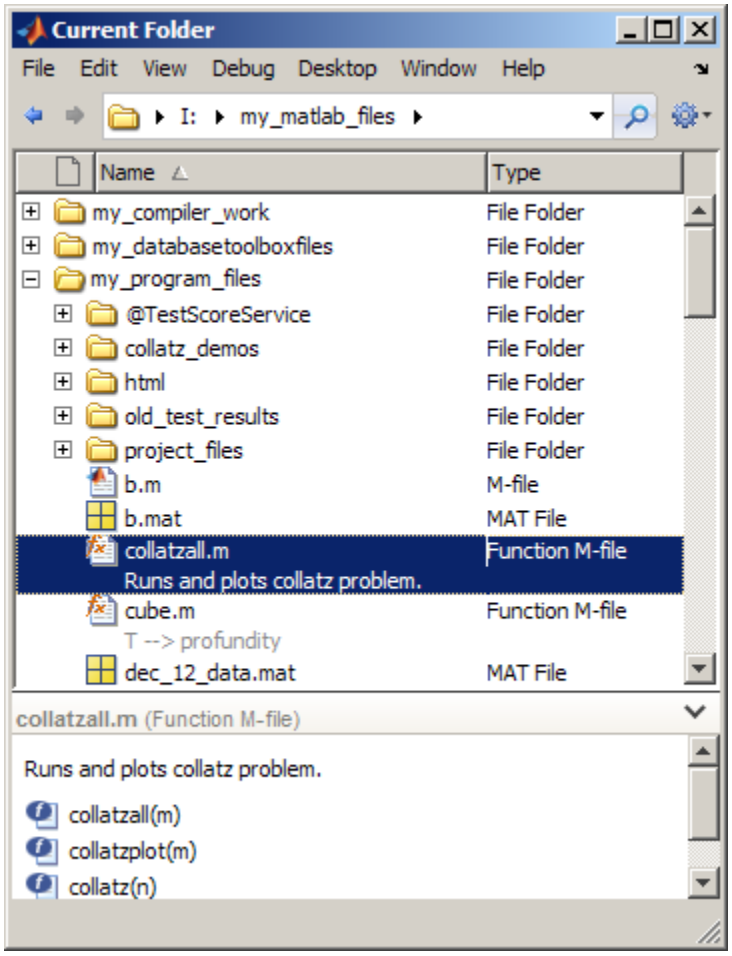
Enhanced Current Folder (Directory) Browser

The Current Directory browser is now the Current Folder browser.

For an overview of the enhancements, watch this video, Current Folder Browser Enhancements, or review the following summary:

- Use the expandable tree structure to view the contents of the current folder. Double-click a subfolder to make it the current folder.
- Display the file type in a column.
- Distinguish types of MATLAB program files using new icons:
  -  class
  -  function
  -  script
- View file descriptions below file names. To show or hide descriptions, select **View > Show > Description**.
- Display hidden files a new way:
  - On Windows platforms, MATLAB follows your Windows preference for hidden files.
  - On other platforms, MATLAB uses the new setting in Current Folder preferences.
- Find files and folders within the current folder and its subfolders:
  - To access the search feature, click the Search button  on the address bar, and then enter the string you want to find.
  - To clear the results and display all files and folders in the current folder, click the Clear button  .

The following illustration shows the new Current Folder browser, including the tree view, new icons for program files, and the search button.



Compatibility Considerations

- With the new tree structure, you now see all files, including those in subfolders. You might not be able to run the files you see in the subfolders. To run a file from a subfolder, the subfolder must be on the search path or the subfolder must become the current folder. In previous versions, you could only see files in the current folder, so you could run every file you could see.
- You now search for files by clicking the search button in the address bar. In previous versions, the search feature, referred to as the filter field, appeared if your preference was set to display it.
- File descriptions now appear below the file name. In previous versions, they displayed in a column.

For more information about the new features, see Managing Files in MATLAB.

File Exchange Desktop Tool — Find and Get Files Created by Other Users

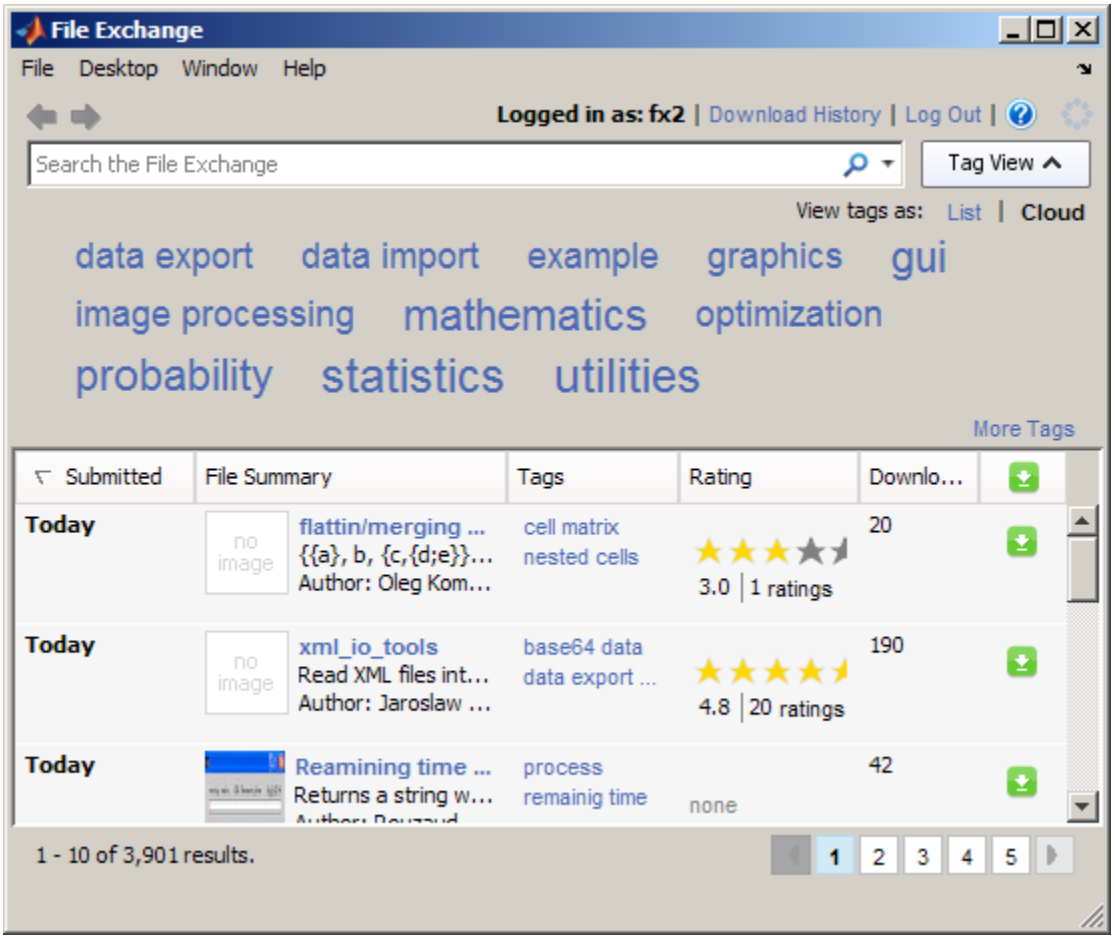
Watch this video, MATLAB File Exchange Access, or review the following summary of what you can do with the new File Exchange desktop tool:

- Access user-created files from the File Exchange repository. The files are at MATLAB Central, the community area of the MathWorks Web site.
- Use the files in your own work to save time and get new ideas.
- Work with the file repository from within the MATLAB desktop.
- Use features like those found in the Web site interface to the File Exchange repository.

The File Exchange desktop tool offers these main features:

- Find files by searching and by selecting tags (keywords associated with files).
- Sort results. For example, show the most highly rated or the most recent files first.
- View details about a file.
- Download files to use in MATLAB.
- Provide feedback about files.

To open the File Exchange tool, select **Desktop > File Exchange**.



If you have questions while you work, access the File Exchange FAQ by clicking the Help button . For full documentation, see File Exchange — Finding and Getting Files Created by Other Users.

Editing and Debugging MATLAB Code Files

New features and changes introduced in Version 7.9 (R2009b) are:

Syntax Highlighting for VHDL and Verilog Code

The MATLAB Editor now supports syntax highlighting for VHDL and Verilog code. For details see, Highlight Syntax to Help Ensure Correct Entries in the Editor.

File and Folders Comparison Tool Enhanced

When you use the File and Folders Comparisons tool to compare folders, it now includes the following information about each file in each folder:

- The file size, in bytes
- The date the file was last modified

In addition, the File and Folders Comparisons tool enables you to reload the information by clicking the refresh button  or selecting **File** > **Refresh**. For details see, Comparing Files and Folders.

Publishing MATLAB Code Files to PDF Output Format

MATLAB now supports PDF as the output format for publishing MATLAB code files. For instructions on how to publish to `.pdf` files, watch the Publishing Features video demo. For details, see Steps for Publishing sine_wave_f.m to PDF.

Internationalization

How MATLAB Reads Customized Locale Settings on Macintosh OS X Platforms

If you use the **Customize** option to modify the locale setting, MATLAB ignores the customized portion. For example, if you select the euro currency symbol for a locale set to the United Kingdom, the locale name is:

```
en_GB@currency=EUR
```

MATLAB uses the locale name:

```
en_GB
```

Mathematics

Computational Geometry Functions Being Changed

2- and 3-D computational geometry functions (`delaunay`, `convhull`, `griddata`, `voronoi`, `delaunay3`, `griddata3`) no longer use QHULL or the QHULL options arguments. The N-D functions `gridatan`, `delaunayn`, `convhulln`, and `voronoin` still use QHULL.

Compatibility Considerations

The QHULL options arguments to `delaunay`, `convhull`, `griddata`, and `voronoi` are no longer required and are currently ignored. Support for these options will be removed in a future release.

Computational Geometry Functions Being Removed

A future release will remove the `griddata3`, `dsearch`, `tsearch`, and `delaunay3` functions. See the table below for alternatives.

Function Name	Use This Method Instead
`griddata3`	`TriScatteredInterp`
`dsearch`	`DelaunayTri/nearestNeighbor`
`tsearch`	`DelaunayTri/pointLocation`
`delaunay3`	`DelaunayTri`

Compatibility Considerations

Update your old code to use the new computational geometry classes `DelaunayTri` and `TriScatteredInterp`.

New Sparse Matrix Functionality In qr and mldivide Functions

`mldivide` supports complex rectangular sparse input matrices. `qr` supports complex sparse input matrices. `qr` for sparse matrix input now supports a third output argument that contains a fill-reducing permutation.

Support for Large-Sized Dimensions In fft

MATLAB functions `fft`, `fft2`, and `fftn` (and their inverses `ifft`, `ifft2`, and `ifftn`) can now handle input arrays with a size in 1 dimension greater than $2^{31} - 1$ on 64-bit platforms.

Performance Improvement For Large Data Sets

- MATLAB includes improved sparse matrix performance for indexing, basic math, binary and relational operators, and exponential functions.
- There are significant performance improvements to `conv2`.

erfcore Being Removed

Use of `erfcore` will produce a warning:

```
ERFCORE will be removed in a future release.
Use ERF, ERFC, ERFCX, ERFINV, or ERFCINV instead.
```

Compatibility Considerations

Replace all instances of `erfcore` with `erf`, `erfc`, `erfcx`, `erfinv`, or `erfcinv`.

New Multithreading Capability

Many MATLAB functions are now multithreaded:

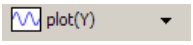
- `sort`
- `bsxfun`
- `mldivide` for sparse rectangular matrix input
- `qr` for sparse matrix input
- `filter` for matrices and higher-dimensional arrays
- `gamma`, `gammaln`
- `erf`, `erfc`, `erfcx`, `erfinv`, `erfcinv`

New Test Matrices in gallery Function

The `gallery` suite of test matrices includes new options `integerdata`, `normaldata`, and `uniformdata`.

Data Analysis

Improved Plot Selector Makes Graphic Data Exploration Easier

The Plot Selector button on the Workspace Browser and Variable Editor toolbars has a new look and added capabilities, options, and help. Now the Plot Selector button 

changes its appearance to reflect the variable or variables you select. Now it suggests a plot type and indicates its calling argument sequence. You can also make the Plot Selector list the types of graphs you use most often at the top of its drop-down menu. The tool provides descriptions and hints to help you discover data graphing functions and learn to use them more effectively.

For further details, see this video demo and the release note “Enhanced Plot Selector Simplifies Data Display” on page 26-19.

Programming

Ignore Selected Arguments on Function Calls

This version of MATLAB introduces a new usage for the tilde (~) operator. As in earlier releases, tilde signifies a logical NOT. In the 9b release, you can also use the tilde operator to specify unused outputs in a function call, or unused inputs in a function definition. See Ignore Function Inputs in the Programming Fundamentals documentation for more information on this feature.

Replacing Output Variables with Tilde

This feature enables you to replace this type of function call:

```
[val1, ignoreThisOutput, val3] = myTestFun;
```

with the following:

```
[val1, ~, val3] = myTestFun;
```

MATLAB ignores any values returned by a function that have a corresponding tilde in the output list. This new syntax can help you avoid confusion in your program code and unnecessary clutter in your workspace. It also avoids wasting memory to store unused outputs returned from the called function.

Replacing Input Arguments with Tilde

You can also use the tilde operator to specify unused inputs when you are creating a function. You can replace this type of function definition:

```
function myTestFun(arg1, ignoreThisInput, arg3)
```

with the following

```
function myTestFun(arg1, ~, arg)
```

Whenever this function is called, MATLAB ignores any inputs passed to the function that have a tilde in the corresponding position in the input argument list. You are likely to find the tilde operator most useful when writing callback functions and subclass methods that must match a predefined function interface.

Internal Packages Make Reserved Functions Easy to Identify

The MathWorks® reserves the use of packages named `internal` for utility functions used by internal MATLAB code. Functions that belong to an `internal` package are intended for The MathWorks use only. Using functions that belong to an `internal` package is strongly discouraged. These functions are not guaranteed to work in a consistent manner from one release to the next. In fact, any of these functions and classes may be removed from the MATLAB software in any subsequent release without notice and without documentation in the product release notes.

See Internal Utility Functions in the Programming Fundamentals documentation for more information.

Use of lasterror, lasterr, rethrow(errStruct) Not Recommended

In version 7.5, The MathWorks introduced a new class called `MException` for use in error handling. Prior to this change, the `lasterr` and, more recently, `lasterror` functions kept track of only the

one most recently thrown error. The state of any errors thrown before that was overwritten by newer errors. Also, this error state was globally accessible, which means that it could be unintentionally modified or destroyed either during an interactive session or by another function.

When using the newer, object-based error mechanism, every error generated by a MATLAB function stores information about what caused the error in a separate `MException` object. This enables MATLAB to store information about multiple errors, adds the capacity to store new fields of information including links to related errors, and also resolves the problem of being globally accessible.

The MathWorks now discourages the use of the `lasterr` and `lasterror` functions because the information they return is global and thus has the potential to be corrupted. You should replace the use of these two functions with the new style of `try-catch` statement that captures an `MException` object to represent the error. When entering commands at the MATLAB command line, you can also use the static method `Mexception.last` to retrieve information on the most recent error. For more information on this method of error handling, see The MException Class.

This change also affects which form of the `rethrow` function to use when reissuing an exception. The original `rethrow` function, in use since before version 7.5, accepts only a structure as input. Because this structure is typically obtained using the `lasterror` function, this form of `rethrow` should be avoided. The later form of `rethrow(MException)`, introduced in version 7.5, accepts an `MException` object as input, and is the recommended form.

Compatibility Considerations

It is strongly recommended that you discontinue the use of `lasterror` and `lasterr`, and that you replace the use of these functions in your program code with the `MException`-based style of error handling described above.

Use of maxNumCompThreads No Longer Recommended

Invoking the function `maxNumCompThreads` now returns the following warning:

```
Warning: maxNumCompThreads will be removed in a future release.
Please remove any instances of this function from your code.
```

`maxNumCompThreads` continues to operate the same in this release of MATLAB. However, the function will be discontinued in a future release of the product.

Compatibility Considerations

It is recommended that you discontinue the use of `maxNumCompThreads`. However, if you do have any program code that invokes this function, you should make sure that such programs are not affected by this new warning.

Excel Worksheet Selection in the Import Wizard

In previous releases, the Import Wizard imported only the first populated worksheet in an Excel file. The Import Wizard now allows you to specify the worksheet to import. To start the Import Wizard, use one of the following methods:

- Select **File > Import Data**.
- Call `uiimport`.
- Double-click a file name in the Current Folder browser.

Motion JPEG 2000 Files Supported by mmreader

`mmreader` now imports Motion JPEG 2000 (`.mj2`) files on Windows, Macintosh, and Linux platforms.

Compatibility Considerations

Because `mmreader` imports Motion JPEG 2000 files, the function `mmreader.isPlatformSupported` always returns `true` on Windows, Macintosh, and Linux platforms. For a list of file formats that `mmreader` supports, and the requirements to read these formats on each platform, see the `mmreader` reference page.

Minimum Sample Rate for audioplayer

If you specify a sample rate less than 80 samples per second, `audioplayer` generates an error with the following ID:

```
MATLAB:audioplayer:positivesamplerate
```

Compatibility Considerations

In previous releases, the error ID on Windows platforms for low sample rates was:

```
MATLAB:audioplayer:negativesamplerate
```

Change any references to this identifier to:

```
MATLAB:audioplayer:positivesamplerate
```

Documentation Changes: File I/O and Data Import and Export

The function category “File I/O” is renamed “Data Import and Export.” It appears immediately below the category “Desktop Tools and Development Environment.”

In the MATLAB User Guide, Data Import and Export content previously appeared in the Programming Fundamentals documentation. The User Guide now includes a stand-alone topic for Data Import and Export.

Object Array Property Indexing

Object array indexing operations on properties now return an error for improper array references. Before MATLAB 7.9, an expression such as:

```
obj.Prop(n) % for non-scalar obj is invalid if Prop is a property
```

did not return an error. MATLAB 7.9, you can reference or assign properties from scalar objects only by entering:

```
obj(int).Prop(n)
```

where `int` is a positive integer.

Equality of Objects Using isequal Now Ignores Numeric Class

Before MATLAB 7.9, an expression such as:

```
isequal(a,b)
```

returned false in cases where `a` and `b` are objects of the same class that have properties set to numeric values which are mathematically equivalent, but of different classes (for example, `double` and `single`),

The behavior of `isequal` in MATLAB 7.9 is consistent with the documented behavior (see `isequal`). `isequal` does not consider class when comparing numeric values.

Class Defining Private/Abstract Property Now Errors

MATLAB 7.9 returns an error when it loads a class that defines a property as both `Abstract` and `Private`.

Subclasses of Built-in Classes and numel

Before MATLAB 7.9, the `numel` function did not return the same results for built-in classes and subclasses of built-in classes. In MATLAB 7.9, the behavior of `numel` is consistent with built-in classes and subclasses of built-in classes. See Understanding size and numel for a discussion of the current behavior.

Array Expansion with Indexed Assignment

Before MATLAB 7.9, initializing a handle object array by specifying the last element in the array to create it:

```
obj(10) = MyClass;
```

resulted in the same handle being assigned from the second to the penultimate elements of the array. With MATLAB 7.9, all elements in the handle object array have unique handles. See Creating Object Arrays for more information on the current behavior.

New Tiff Object Enables Writing of Tiled Data and Broader Metadata Support

Using the new `Tiff` object, you can now write portions of a TIFF file and update the values of individual metadata tags in an image. MATLAB has long offered the capability of reading and writing TIFF files, using the `imread` and `imwrite` functions. However, if you wanted to update any part of the image, you had to write the entire image to the file. Similarly, if you just wanted to update a tag, you had to write the entire image. By providing access to functions in the LibTIFF library, the Tiff object offers more flexibility in creating and editing TIFF images. You can write data to specific tiles in an image and update individual tags in a file, without having to write the entire image.

Ambiguity Error Now Reported

Due to a bug, versions of MATLAB prior to Version 7.9 did not report an error in functions such as the following, even though it cannot be determined in the last line whether `U` is a variable or a function:

```
function r = testMfile1(A)
A = 1;
eval('U = 1;');
r = A(logical(end), U(end));
```

However, if you replace the last line with

```
r = A(U(end));
```

MATLAB reports an ambiguity error and has done so since Version 7.0. Starting in Version 7.9, MATLAB reports an ambiguity error for either statement.

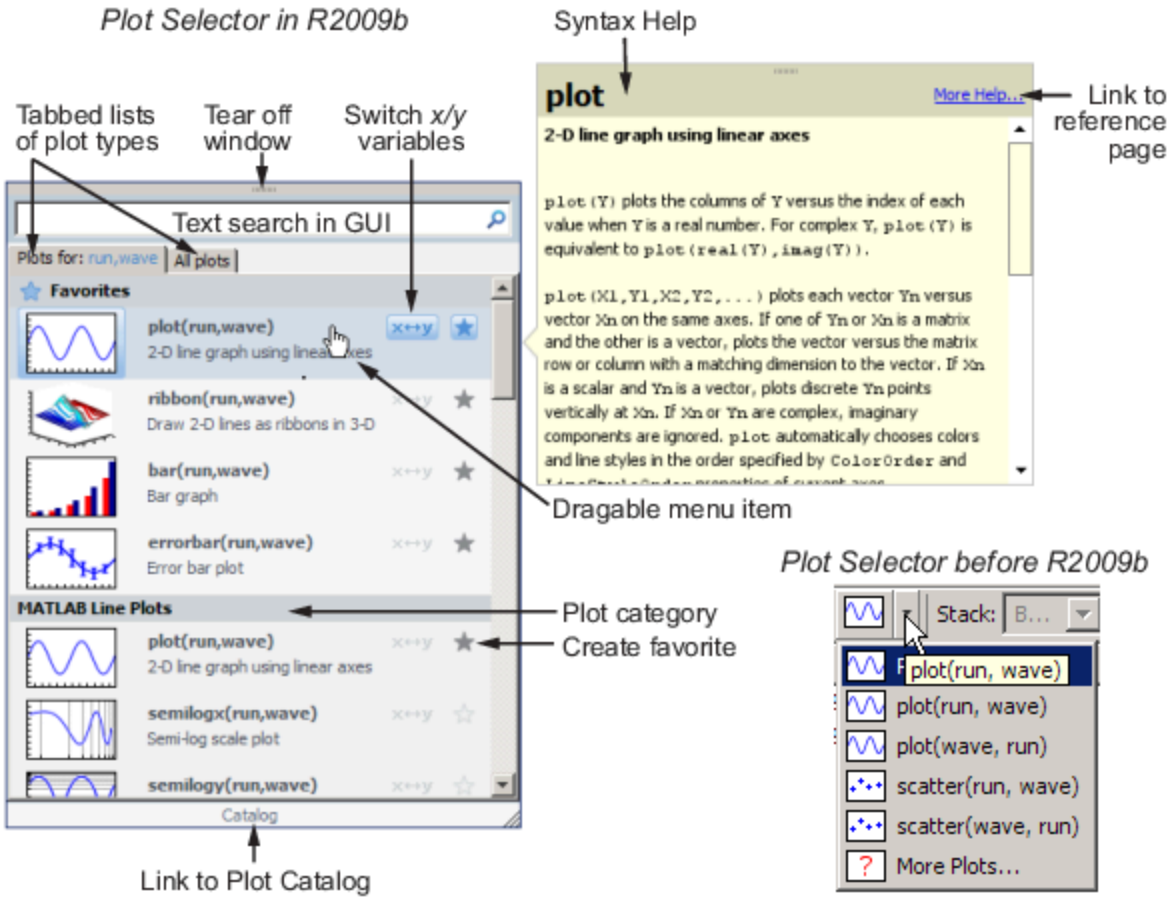
Compatibility Considerations

In certain circumstances, it is possible that you will see ambiguity errors generated in code that did not report this error in previous releases. If you do get such an error, examine your code and resolve the ambiguous statements.

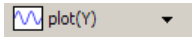
Graphics and 3-D Visualization

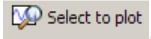
Enhanced Plot Selector Simplifies Data Display

The Plot Selector workspace tool creates graphs of workspace variables. As this video demo shows, the tool lets you access more types of data display functions and provides help about each one. It also categorizes display functions and lets you organize them within its drop-down menu. The following illustration shows the old and new versions of the Plot Selector and calls out new features.



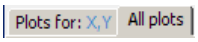
Use the Plot Selector tool to instantly generate graphs of workspace variables. You can choose from more than 40 two-dimensional, three-dimensional, and volumetric MATLAB data display types. It also constructs graphs using Control System Toolbox, Financial Toolbox, and Statistics and Machine Learning Toolbox functions if your installation includes those products.

The Plot Selector button 

displays a graph icon and the names of selected variables in the Workspace browser or the Variable Editor. Until you select one or more variables, the button reads **Select to plot** 

. The graphing function for one or two data vectors still defaults to `plot`, but the tool now gives you additional options and information, including the following:

- The grouping of menu items into categories of graph types, such as **Stem and Stair Plots**, **3-D Surface Plots**, and **Analytic Plots**. You can rearrange categories and items within a category by dragging them.

- A **Favorites** category at the top of the menu, where you can collect the types of graphs you use most often. Your collection of favorites persists across MATLAB sessions.
- A star-shaped button  on menu items that adds a graph type to or removes one from your Favorites collection.
- Two tabs 

for toggling the scope of the menu:
  - **Plots for <variable names>** — The set of graphs you can generate with the currently selected variables
  - **All plots** — A master list of graphing functions available on your system

  On the **All plots** tab, graphing functions that are not compatible with the selected variables display in gray and do not plot. Incompatible plot types do not appear on the first-tab menu.
- A button 

to interchange the axes positions of two selected variables
- Pop-up descriptions of function syntax when your mouse pointer lingers over a menu item
- Direct access to your Favorites from context menus in the Workspace Browser and the Variable Editor

For more information, see Creating Plots from the Workspace Browser.

Compatibility Considerations

- In previous versions, the Plot Selector automatically assigned variables named `t` and `time` to the *x*-axis. Now, the variable you select first always plots on the *x*- axis. Use the switch variable input order button 

to interchange the *x* and *y* variables before plotting them.
- In previous versions, clicking **Plot Selector** > **More Plots** opened the Plot Catalog tool. Now, you click **Catalog** at the bottom of the Plot Selector GUI to open the Plot Catalog tool. The Plot Catalog has not changed.

Certain Print Options and Devices Now Warn When Used

The following print command option and device now throw warnings when used and will be removed in a future release:

Option/ Device	Warning ID	Warning Text
`'-adobecset'` print option	`'MATLAB:print: adobecset:DeprecatedOption'`	The `-adobecset` option will be removed in a future release.
`'-dill'` print device (native support for Adobe Illustrator)	`'MATLAB:print: Illustrator:DeprecatedDevice'`	The `'-dill'` print device will be removed in a future release. Use Encapsulated PostScript (`'-depsc'`) instead.

If you wish to prevent these warnings from displaying, use the warning command.

- To disable the warning generated by using the `–adobecset` print option:

```
warning('off', 'MATLAB:print:adobecset:DeprecatedOption');
```

- To disable the warning generated by using the `-dill` print device:

  ```
  warning('off', 'MATLAB:print:Illustrator:DeprecatedDevice');
  ```

Compatibility Considerations

If you have developed code that uses the previous option or device, in the future you must remove such calls or replace them with other code.

The view Function No Longer Supports 4-by-4 Transformation Matrices as Input

The `view` function. no longer supports 4-by-4 transformation matrices as input arguments. Previously, you could create a 4-by-4 transformation matrix and use it with `view`.

Compatibility Considerations

If you have code that calls `view(T)`, explore using `view([az el])` instead.

Creating Graphical User Interfaces (GUIs)

Expanded Documentation on Techniques for Programmatic GUI Design

The section Lay Out a Programmatic GUI of the Creating Graphic User Interfaces documentation has been expanded. The updated section, formerly called “Aligning Components,” now has the title Compose and Code GUIs with Interactive Tools.

The renamed section describes techniques, functions, and predefined dialog boxes you can use to accelerate programmatic construction of GUIs (that is, those you create without using GUIDE).

The section includes examples of setting component positions, colors, and font properties interactively. Also included is an example function that generates a `set` statement for a specified property of any GUI component. By running this function and pasting its output into your GUI code file, you can save time and avoid making typographical errors.

Previous Change to How UI Components Set the Figure SelectionType Property

The MATLAB `SelectionType` figure property had an undocumented change in V. 7.6 (R2008a). That change modified how the property is set in response to single-clicking and double-clicking objects, particularly uicontrols, with and without key modifiers.

For information about this change, including potential incompatibilities, see “Changes to How uicontrols Set Figure SelectionType” on page 29-33. The table in that release note specifies the `SelectionType` setting resulting from clicking a UI component in both enabled and disabled states.

External Interfaces/API

Changes to Compiler Support

Support for Apple Macintosh (64-bit) Platforms

MATLAB Version 7.9 (R2009b) supports these new compilers for building MEX-files:

- Apple Xcode 3.1 (gcc / g++ Version 4.0.1)
- GNU gfortran Version 4.3

Compiler Support to Be Phased Out

MATLAB Version 7.9 (R2009b) supports the following compilers but will not support them in a future version.

Windows (32-bit) Platforms

- Intel Visual Fortran Version 10.1
- Intel C/C++ Version 9.1
- Microsoft Visual Studio .NET Version 7.1

Windows (64-bit) Platforms

- Intel Visual Fortran Version 10.1
- Intel C/C++ Version 9.1

Solaris SPARC (64-bit) Platforms

- Sun Studio 11 cc / CC Version 5.8
- Sun Studio 11 f90 Version 8.2

Discontinued Support for Intel Visual Fortran Version 9.1

MATLAB no longer supports the Intel Visual Fortran Version 9.1 compiler on the following platforms:

- Windows 32-bit
- Windows 64-bit

Compatibility Considerations

To ensure continued support for building your Fortran programs, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Run-Time Libraries Required for Applications Built with Microsoft Visual Studio 2008 Compiler

You must provide the Visual C++ run-time libraries if you distribute any of the following:

- MEX-file

- Engine application
- MAT-file application built with the Visual Studio 2008 compiler

You need these files to run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed. For information on locating the Microsoft Visual C++ 2008 Redistributable Package (x86), containing `vcredist_x86.exe` and `vcredist_x64.exe`, consult your Microsoft documentation.

Changes to Building MEX-Files

INLINE Option to MEX Function Deprecated

The use of the `-inline` switch has been deprecated and will not be available in future versions of the `mex` function.

MEX Function No Longer Automatically Includes mexversion.c When Building MEX-Files

The `mex` function no longer automatically compiles and links its own copy of `mexversion.c` when generating MEX-files.

New Features for Interface to Microsoft .NET Framework

This release includes several changes that affect your use of the Microsoft .NET framework:

- The `NET.addAssembly` function returns information about the assembly in the `NET.Assembly` class. You can also add an assembly using an instance of the `System.Reflection.AssemblyName` class.
- You can access elements of a .NET array using MATLAB one-based indexing, as described in Accessing .NET Array Elements in MATLAB.
- To call a generic method, use the `NET.invokeGenericMethod` function.
- You can change a static property or field name using the `NET.setStaticProperty` function.
- You can use overloaded operators, such as `+` and `*`. For a complete list of supported operators, see How MATLAB Represents .NET Operators.

R2009a

Desktop Tools and Development Environment

Desktop New Features Video

For an overview of the major new features in the MATLAB Desktop Tools and Development Environment area, watch this video demo. Another way to access the demo is by selecting the **Demos** tab in the Help browser, and then selecting **MATLAB > New Features in Version 7.8**.

Startup and Shutdown

The `matlab` command to start MATLAB now supports the `-singleCompThread` option on all platforms. When you specify this option, you limit MATLAB to a single computational thread. By default, if you do not specify this option, MATLAB makes use of the multithreading capabilities of the computer on which it is running.

Desktop

New features and changes introduced in Version 7.8 (R2009a) are:

New System Browser Preference Instead of docopt.m for MATLAB on UNIX Platforms

On UNIX platforms (except Apple Macintosh), MATLAB now uses the Mozilla® Firefox browser by default to display documents or Web sites in a system browser. If you want MATLAB to use a different system browser, use the new **System Web browser** setting in Web preferences to specify it. In addition, the `web` function with the `-browser` option now determines the system browser to use from the preference. For more information, click the **Help** button in the Web preference pane, or see Web Preferences.

Compatibility Considerations

In previous versions, the default system browser on UNIX platforms was Netscape Navigator®; it is now Firefox. If you do not have Firefox on your system, when MATLAB tries to use a system browser, it produces a warning. To correct the problem, use Web preferences to specify a system browser that is installed.

In previous versions, if you wanted to use a different browser, you specified it in the `docopt.m` file. Starting in R2009a, MATLAB ignores the browser specified in `docopt.m`. If you have code that relies on `docopt.m`, your code still runs, but it produces a warning. Remove the calls from your code. In future versions, the code will not run and will produce an error.

If you have your own `docopt.m` file, delete it and either use the new default, Firefox, or specify a different system browser using Web preferences.

Test Proxy Settings for Accessing the Internet from MATLAB

If you want to access the Internet from MATLAB, and your network uses a firewall or another means of protection that restricts Internet access, you now can ensure your proxy server settings are

working correctly. Click the new **Test Connection** button in Web preferences, and MATLAB will use the proxy settings you specified in Web preferences to attempt to access the Internet. For more information about the proxy server features, click the **Help** button in the Web preferences pane, or see Web Preferences.

Running Functions — Command Window and History

Tab Completion for Class Directories and File Names

The Command Window and Editor now support tab completion for class directories. In addition, the Command Window supports tab completion for class file names. For details, see Complete Names in the Command Window Using the Tab Key.

Help and Related Resources

New features and changes introduced in Version 7.8 (R2009a) are:

- “Help Browser No Longer Reopens at Startup” on page 27-3
- “docsearch Accepts Multiple Words Without Parentheses” on page 27-3
- “View Your Platform (32-bit or 64-bit) and Architecture in the About Dialog Box” on page 27-3

Help Browser No Longer Reopens at Startup

When you start MATLAB, the Help browser no longer automatically opens if you had it open when you last quit MATLAB.

Compatibility Considerations

In previous versions, the Help browser opened at MATLAB startup if it had been open when you last quit MATLAB. If you want the Help browser to automatically open at startup, use a startup option. For example, you can include a `helpbrowser` statement in your `startup.m` file. For more information, see Startup Options.

If you include a statement in a `finish.m` file that closes the Help browser automatically whenever you quit MATLAB, you now can remove that statement because MATLAB now performs the action by default.

docsearch Accepts Multiple Words Without Parentheses

The `docsearch` function, which you can use to search the documentation, now accepts multiple words as input, without requiring the function form of the syntax. For example, in previous versions, you used `docsearch('word1 word2')`, but now you can use `docsearch word1 word2`. With the new form of the syntax, you can use all options for `docsearch`, such as wildcards (for example, `docsearch wor*`) and exact phrases (for example, `docsearch "word1 word2"`).

View Your Platform (32-bit or 64-bit) and Architecture in the About Dialog Box

To find out if you are currently running a 32-bit or 64-bit version of MATLAB, select **Help > About MATLAB**, and view the value in the resulting About MATLAB dialog box. You might need to know the version if you want to take advantage of the benefits of 64-bit MATLAB, or if you want to use files that depend on the version, such as MEX-files.

The About MATLAB dialog box also shows the architecture value, for example, `(win32)` or `(win64)`. You use this value for the `arch` option of the `mex` function.

For more information, see Information About your Installation.

Workspace, Search Path, and File Operations

Enhancements to Current Directory Browser

These are the enhancements to the Current Directory browser:

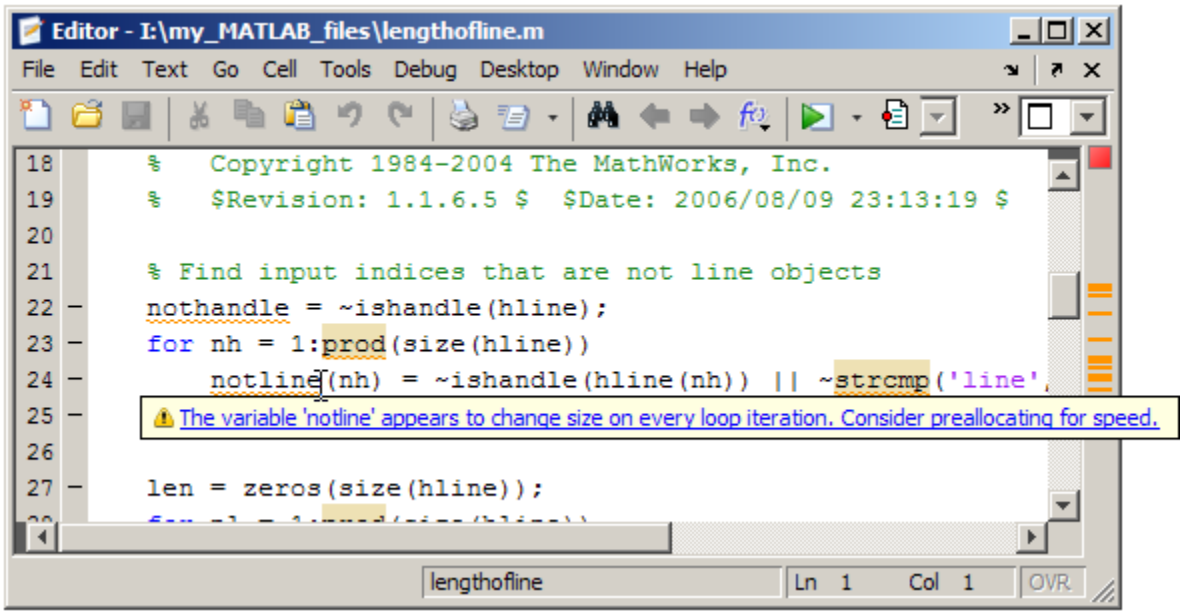
- You can use the new **View** menu to choose the columns to display, to specify the sort order for a column, and to apply grouping for any attribute. In the previous version, you performed these actions using the column header and its context menus, which you can still do (but only on Microsoft Windows platforms). For more information, see Sorting and Grouping Files and Folders.
- A new **Description** column displays the brief description for files and directories in the current directory. To show the column, use **View > Choose Columns**. Descriptions include the first help line in a MATLAB program file, and a model file's description, which is useful because you do not need to start the Simulink software to view it. The description is the same one that appears with the details for a selected file. For more information, see Viewing Help for a MATLAB Program File.
- If you try to rename a MATLAB program file in the Current Directory browser to an invalid name, such as `*myfile.m`, a warning appears, notifying you about the name problem. If you want to run the file, change the file name to a valid one. For more information about what a MATLAB file name requires to be valid so you can run it, see Naming Functions.
- When you right-click a FIG-file in the Current Directory browser, there is a new option to open the figure in GUIDE.

Editing and Debugging MATLAB Code

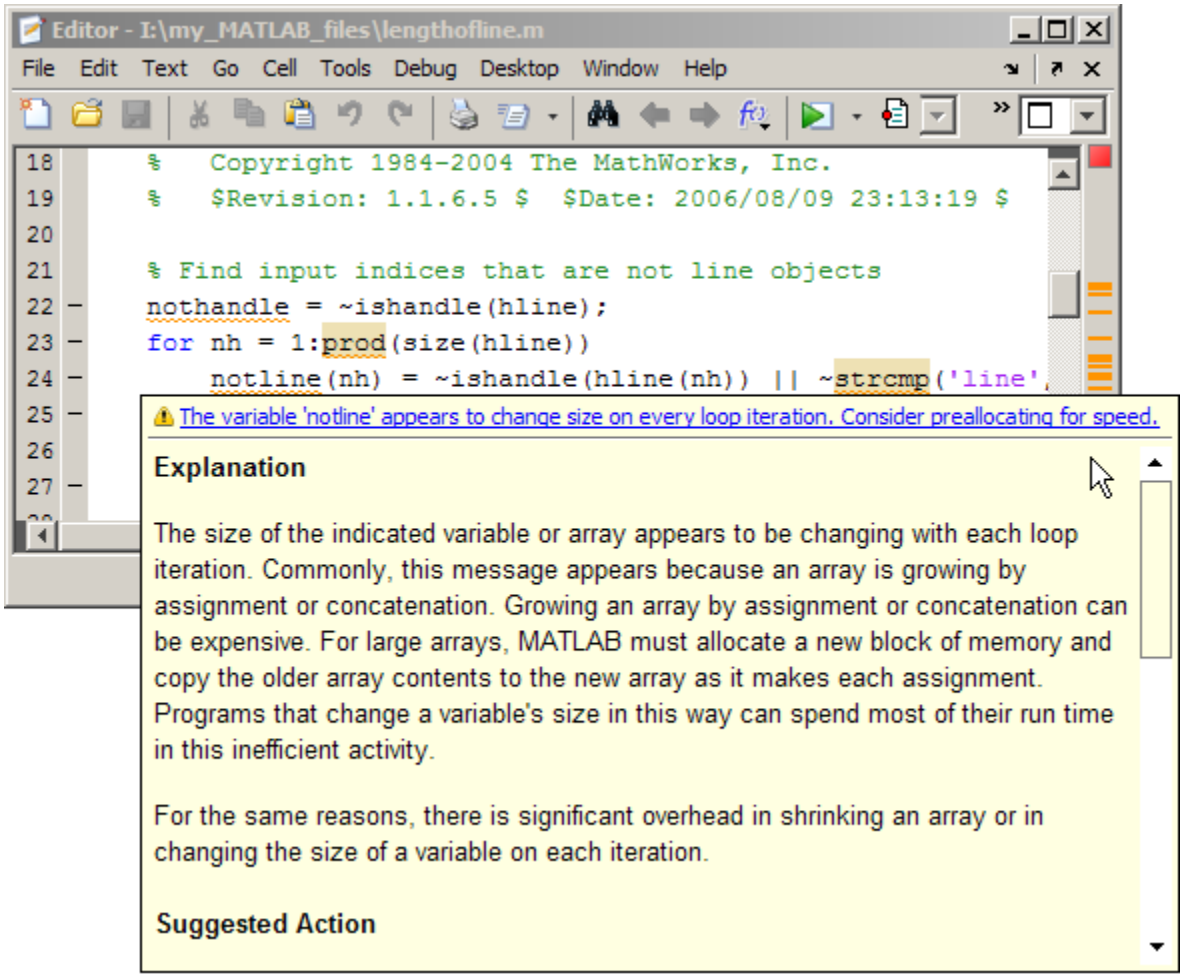
New features and changes introduced in Version 7.8 (R2009a) are:

Many M-Lint Messages Now Extend to Provide an Explanation and Suggested Action

When you create a file with integrated M-Lint warning and error messages enabled, you can get additional information about many of the messages. When you hover the pointer over an M-Lint indicator that has an extended message, the message appears as a link:



When you click the link, the message window expands to display an explanation and suggested action.



When you click a link within the extended message, the Help browser opens to provide more information.

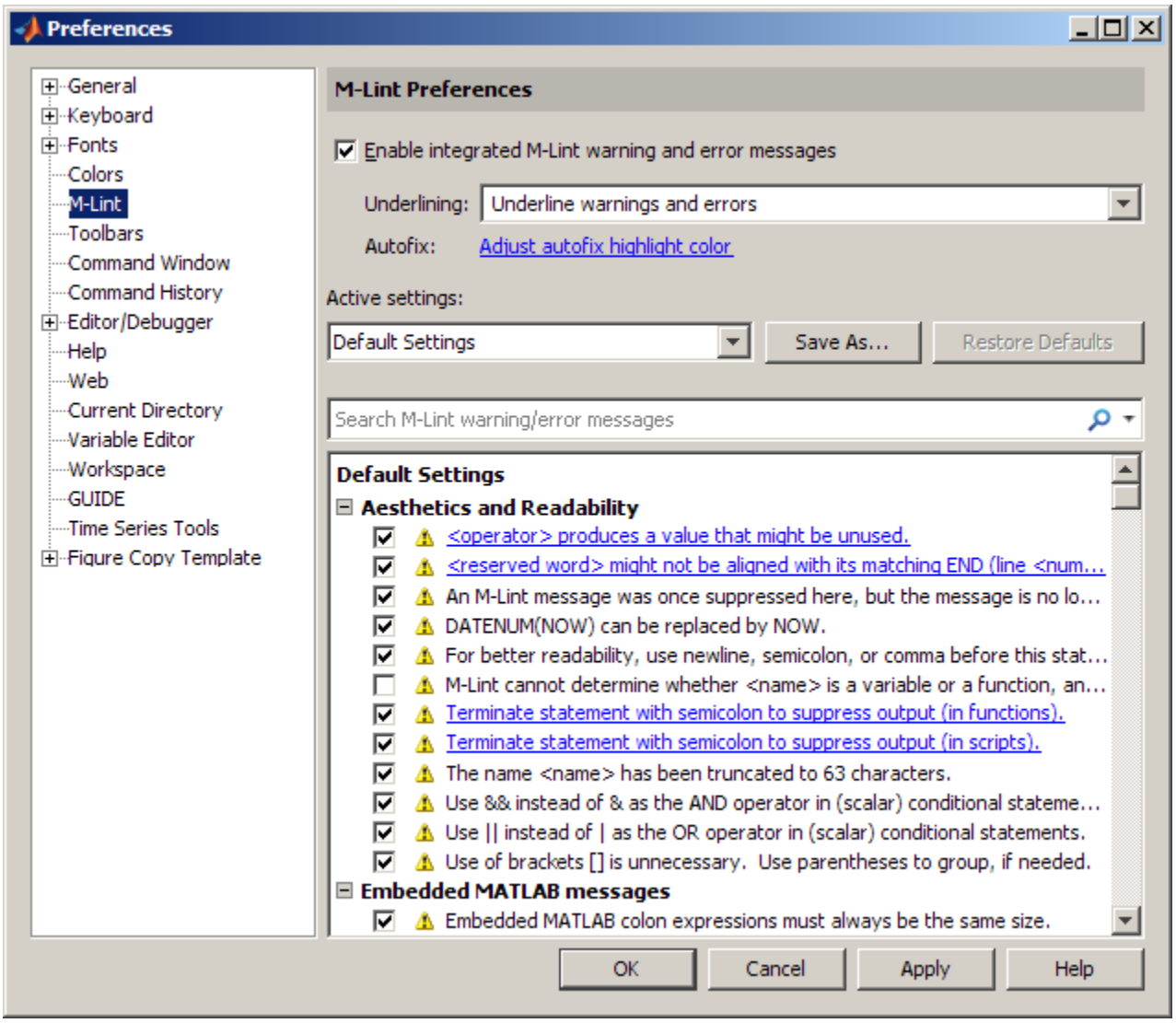
M-Lint Messages Now Searchable in Preferences

You can now filter the list of M-Lint messages in the preferences panel (**File > Preferences > M-Lint**), to find a message of interest.

For example, you can search for a message:

- Containing a string
- Corresponding to a particular message ID
- Within a given category
- With a setting different from the default

Filtering the list of messages can help you see, for example, why certain messages are suppressed, and which messages are disabled. It can be helpful when you want to see the explanation and suggested action for a message, as described in “Many M-Lint Messages Now Extend to Provide an Explanation and Suggested Action” on page 27-4.



Block Indenting Option No Longer Provided

The **Block Indent** option is no longer available. Previously, this option was available for MATLAB, Java, and C/C++ programming languages, when you selected **File > Preferences > Editor/Debugger > Language**.

Note Do not confuse the **Block Indent** option with the **Smart indenting** option, which is still provided.

Compatibility Considerations

To attain the effect of block indenting, you can use the **No indent** option and indent lines manually using the **Tab** and **space** keys.

Integrated Text Editor Option Removed from Editor/Debugger Preferences Panel

The MATLAB Editor no longer supports EmacsLink. In previous releases, you could choose **File > Preferences > Editor/Debugger**, and then (if you correctly registered EmacsLink with MATLAB) you could select **Integrated text editor**. This option is no longer available.

New Navigation Aids in File and Directory Comparisons Tool

The File and Directory Comparisons Tool now provides links to help you quickly navigate to the areas of differences. This is useful for large files that have too many lines to fit on the screen. At the top of the page is a link to the first difference. In addition, each set of differences has an up arrow that you click to go to the previous set of differences, and a down arrow you click to go to the next set of differences. For details, see Stepping Through Differences.

Wrap Around Option for Find and Replace Now On By Default

When the **Find Next** operation for searching files in the Editor reaches the end of a file, it automatically wraps around to search from the beginning of the file for the text you specified. In previous releases, the **Wrap around** option was off by default. This meant that if you were in the middle of a file and the text you were searching for appeared before your current position, **Find Next** would not find that text.

Because it is more convenient to have this option on, it is now selected by default. To access this option, select **Edit > Find and Replace**.

For more information, see Find and Replace Text in Files.

Tuning and Managing MATLAB Code Files

Profile Summary Report Includes Information on Excluded Profiling Overhead

The bottom of the Profile Summary Report now indicates the amount of time spent in profiling overhead, when possible. For details, see Profile Summary Report.

Publishing MATLAB Code Files

New features and changes introduced in Version 7.8 (R2009a) are:

- “New Options for Capturing Figures in Published Documents” on page 27-8
- “Dynamic Links in Published Documents” on page 27-9

New Options for Capturing Figures in Published Documents

Two new options are available to specify how you want figures captured for published documents: `entireGUIWindow` and `entireFigureWindow`. The `entireGUIWindow` option is appropriate for most publishing purposes and is now the default. The `entireFigureWindow` option is appropriate when you want to capture all the details, including the title bar and other window decorations in your published document. Use this option, for example, if you are creating a tutorial on using MATLAB software. In the GUI, you can select these options from the **Figure capture method** setting in the Edit M-file Configuration dialog box. In the `publish` function, you can specify these options with `figureSnapMethod`.

For more information, see Figure capture method and the `publish` reference page.

Dynamic Links in Published Documents

When you publish a document to HTML, you can include dynamic links. Dynamic links are links to files on the MATLAB path. They are called dynamic links because MATLAB evaluates them when the reader of your document clicks on one. To use a dynamic link, the reader must open the HTML file in the MATLAB Web browser.

For more information, see Dynamic Hyperlinks.

Mathematics

Upgrade to Computational Geometry

MATLAB includes a new object-oriented suite of computational geometry tools, together with a new underlying library called CGAL. The new library provides improved robustness, performance, and memory efficiency. The new tools are presented in three classes:

- New class `TriRep` provides topological and geometric queries for triangulations in 2-D and 3-D space.
- New class `DelaunayTri` provides increased functionality for Delaunay triangulation including topological and geometric queries, incremental modification, and edge constraints.
- New class `TriScatteredInterp` provides fast robust scattered data interpolation and a new natural-neighbor interpolation technique.

Incomplete Inverse Gamma Function gammaincinv and Incomplete Inverse Beta Function betaincinv

MATLAB has new functions for incomplete inverse gamma and incomplete inverse beta functions. These functions, `gammaincinv` and `betaincinv`, provide the Statistics and Machine Learning Toolbox functionality of the inverse incomplete gamma and beta functions to MATLAB.

Krylov Subspace Methods bicgstabl and tfqmr

MATLAB has new iterative methods for solving systems of linear equations. `bicgstabl` provides the stabilized biconjugate gradients method. `tfqmr` provides a transpose-free implementation of the quasi-minimal residual method.

New Function quad2d

The `quad2d` function provides additional quadrature functionality for nonrectangular areas of integration.

Changes To conv

The `conv` function now accepts the `shape` parameter as input.

Changes To conv2 and convn

Use of the `conv2` and `convn` functions with one empty input now returns the matrix of the correct size as described by the `shape` input.

Compatibility Considerations

Use of the `conv2` and `convn` functions with one empty input no longer returns an empty matrix. Instead, it returns the matrix of size specified by the `shape` input. For information on how to use the `shape` input, see the reference pages for the `conv` and `conv2` functions.

nextpow2 Changing to Element-By-Element Calculation in a Future Release

Compatibility Considerations

`nextpow2` with a nonscalar produces a warning:

```
Warning: NEXTPOW2(X) where X is non-scalar will change
behavior in future versions: it will operate on each element
of X. To retain current behavior, use NEXTPOW2(LENGTH(X)) instead.
```

This behavior will change in a future release. Replace instances of `nextpow2` with nonscalar input `X` to `nextpow2(length(X))` to maintain the current behavior.

Function finite Being Removed

The `finite` function is obsolete. Use of the finite function now causes an error in MATLAB.

Compatibility Considerations

Replace all instances of `finite` with `isfinite`.

New Multithreading Capability in MATLAB Functions

The MATLAB functions for Fourier transforms `fft`, `fft2`, and `fftn`, and their inverses `ifft`, `ifft2`, and `ifftn` are now multithreaded. In addition, the MATLAB functions `prod`, `sum`, `max`, and `min` are multithreaded.

64-bit Support in LAPACK and BLAS

MATLAB supports 64-bit integers for matrix dimensions in LAPACK, and BLAS. Linear algebra operations can now handle matrices of dimensions greater than $2^{31} - 1$.

Compatibility Considerations

`MEX` files that call BLAS or LAPACK need to be updated. All integer variables passed into BLAS or LAPACK need to be of type `mwSignedIndex`. `MEX` files compiled in previous versions of MATLAB that call BLAS or LAPACK could lead to undefined behaviors. If you have existing code that generates `MEX` files that pass variables to BLAS or LAPACK you need to update the code to use the proper data types and recompile.

Upgrade to ACML 4.1.0

AMD Core Math Library (ACML) is upgraded to version 4.1.0.

Data Analysis

Programming Fundamentals

Setting the Number of Threads Removed from Preferences Panel

The capability to adjust the number of computational threads in a single MATLAB session is no longer available as of this release. This change removes from the MATLAB preferences panel the ability to set the maximum number of computational threads. The primary reason for this change is that products that MATLAB is dependent upon have the ability to spawn threads from the currently-executing thread. This makes it infeasible to monitor and/or limit the number of computational threads at any given time in a MATLAB process.

MATLAB versions 7.8 and later require that you decide between threading and no threading at the time you launch your MATLAB session. Multithreading is enabled by default. To disable this feature, start MATLAB using the new `singleCompThread` option.

Compatibility Considerations

If you currently use the preferences panel to enable or disable multithreading or to adjust the number of computational threads, you need to be aware that this capability is no longer available. See the Startup Optionssection in the Desktop Tools and Development Environment documentation to find out how to enable or disable multithreading when launching a new MATLAB session.

Timer Objects Saved in New Format

The format in which MATLAB saves Timer objects has changed in MATLAB version 7.8. Any Timer objects that you create and save while running MATLAB 7.8 cannot be loaded into an earlier version of MATLAB.

Compatibility Considerations

If you need to use a Timer object that you have constructed using MATLAB 7.8, you will have to reconstruct and save the object in an earlier version of MATLAB.

mmreader Supports Linux Platforms

The `mmreader` object now supports Linux platforms. For more information about using `mmreader`, see the `mmreader` reference page.

Support of Microsoft Excel 2007 File Formats

If you have installed Excel 2007 (or Excel 2003 with the Compatibility Pack) on your Windows system, the `xlswrite` function exports data to XLSX, XLSB, and XLSM formats.

To write data to an Excel file, specify the name and extension of the output file in the call to `xlswrite`. If the file already exists, `xlswrite` writes data in the existent file format. If the file does not exist, `xlswrite` creates a new file, using the format that corresponds to the file extension you specify. If you do not specify a file extension, `xlswrite` applies the XLS extension, and writes a new file in the XLS format.

The `xlsread` function imports any file format recognized by your version of Excel, including XLS, XLSX, XLSB, XLSM, and HTML-based formats. The `importdata` function imports XLS, XLSX, XLSB, and XLSM formats. The Import Wizard imports XLS and XLSX formats.

Note Large files in XLSX format might load very slowly. For better import and export performance, Microsoft recommends that you use the XLSB format.

Anonymous Functions Support str2func

The `str2func` which, prior to this release, converted a function name to a function handle, now also converts an anonymous function definition to a function handle. See the function reference page for `str2func` for more information.

```
N = 5;   NthPower = str2func(['@(x)x.^', num2str(N)]);
NthPower(8)
ans =
     32768
```

size and range Implemented for validateattributes

The `validateattributes` function now enables you to check the size and range of the input value. The following commands validate the size and range of the values of `x` and `y` respectively:

```
x = rand(4,2,6);
y = uint8(50:10:200);

validateattributes(x, {'numeric'}, {'size', [4,2,6]});
validateattributes(y, {'uint8'}, {'>=', 50, '<=', 200})
```

isempty Supported for Map Objects

MATLAB now supports the `isempty` function for `containers.Map` objects. This example creates a 0-by-1 Map object and runs `isempty` on it:

```
mapObj = containers.Map;
size(mapObj)
ans =
     0     1

isempty(mapObj)
ans =
     1
```

Bug Fix for Misinterpreted Variables

The example below describes a bug in the MATLAB software that has been fixed in version 7.8. The bug was caused by misinterpretation of an unassigned variable. For certain names, MATLAB mistakenly interpreted the variable as a stem in dot indexing.

The following example illustrates the problem. The example is a bit artificial (which is why the bug went undiscovered for so long).

Suppose you have a function that is intended to perform two levels of dot-indexing:

```
function y = dotindextwice_A(j)
     y = j.lang.String;
```

Calling this function with no arguments results in an error in all versions of MATLAB, as it should.

Now modify the function slightly so that the input variable is named `java` and run it on a version of MATLAB prior to Version 7.8:

```
function y = dotindextwice_B(java)
y = java.lang.String;
```

Now when you run the function without arguments, MATLAB misinterprets the word `java`, treating it as if it were the stem of a Java class name:

```
x = dotindextwice_B;    % Deliberately called with no arguments
```

Prior to Version 7.8, this function did not throw an error. In fact, it returned an instance of the `java.lang.String` class:

```
class(x)
ans =
    java.lang.String
```

This violates the rule that variables in functions are supposed to hide all other uses of the name. Beginning in Version 7.8, calling function `dotindextwice_B` without arguments results in an error, just as calling `dotindextwice_A` does.

MATLAB Upgrades Support for HDF5 to Version 1.8.1

The R2009a release of MATLAB uses version 1.8.1 of the HDF5 library. The HDF Group has deprecated two of the HDF5 library functions, `H5Pget_cache` and `H5Pset_cache`. Their M-file counterparts, `H5P.get_config` and `H5P.set_config`, may not work as they did in prior releases of MATLAB. To replace these deprecated functions in your code, consider these four new HDF5 functions: `H5P.get_mdc_config`, `H5P.set_mdc_config`, `H5F.get_mdc_config`, and `H5F.set_mdc_config`.

Compatibility Considerations

If your code uses `H5P.get_cache` or `H5P.get_cache`, your program will produce a warning message.

Indirect Calls to Superclass Constructors Now Errors

You can no longer call an indirect superclass constructor from a subclass constructor. For example, suppose class B is derived from class A, and class C is derived from class B. The constructor for class C should not call the constructor for class A to initialize properties. The call to initialize class A properties should be made from class B.

Compatibility Considerations

If you define classes in which subclass constructors call indirect superclass constructors, MATLAB now issues an error when you attempt to create an instance of the subclass. Call Only Direct Superclass from Constructor for information on how to correctly code subclass constructors.

Graphics and 3-D Visualization

Functions Previously Only Available in the Image Processing Toolbox Now Available in MATLAB

The following functions have been added to MATLAB from the Image Processing Toolbox:

- `rgb2ind`: Convert RGB image to indexed image.
- `dither`: Convert image using dithering.
- `cmunique`: Eliminate unneeded colors in colormap of indexed image.
- `cmpermute`: Rearrange colors in colormap.
- `imapprox`: Approximate indexed image by one with fewer colors.

Compatibility Considerations

- The `dither` and `imapprox` functions no longer display their output as an image via a call to `imshow` when called with no output arguments. Instead, the first output argument appears in the Command Window if no semicolon ends the line.
- Function `rgb2ind` errors when called with syntax `rgb2ind(rgb)`.
- Function `imapprox` errors when called with syntax `imapprox(x,map)`.

Creating Graphical User Interfaces (GUIs)

New Programmatic GUI Doc Example

A new example in the documentation teaches you how to code a GUI that manages multiple lists, such as to-do and shopping lists, contact information, music or video catalogs, or any set of itemizations. Among other things, it illustrates how to write MATLAB code to:

- Share callbacks among uicontrols.
- Obtain component handles from their tags.
- Create a new version of an existing GUI.
- Import and export list data.
- Edit, add, delete, and reorder list items.
- Save a list in the GUI itself.
- Concurrently run multiple GUIs containing different data that call back to the same function.

This example does not use GUIDE. To read about and run it, see A GUI That Manages List Data in the documentation for Creating Graphical User Interfaces.

GUIDE Help Menu Enhanced

The Help menu in GUIDE now links to more topics in the documentation than previously. It also links to a set of video tutorials on the MATLAB Central Web site. For details, see Getting Help in GUIDE in the Opening GUIDE section of the Creating Graphical User Interfaces documentation.

External Interfaces/API

New Interface to Microsoft .NET Framework

The interface to .NET allows you to bring .NET assemblies into the MATLAB environment, to construct objects from classes contained in the assemblies, and to call methods on these objects. For complete documentation of this feature, see Using .NET Libraries from MATLAB. For an overview of the .NET interface, watch this video demo.

Expanded Platform Support Added for MATLAB Serial Port

MATLAB Serial Port is now supported on the following platforms:

- Microsoft Windows 64-bit
- Apple Macintosh OS X
- Macintosh OS X 64-bit
- Linux
- Linux 64-bit
- Sun Solaris™ 64-bit

Changes To Compiler Support

New Compiler Support

MATLAB Version 7.8 (R2009a) supports these new compilers for building MEX-files:

Microsoft Windows (64- and 32-bit) Platforms

- Microsoft Visual Studio 2008 SP1

Linux (64- and 32-bit) Platforms

- gcc Version 4.2.3

Apple Macintosh (32-bit) Platforms

- GNU gfortran Version 4.2.2

Compiler Support To Be Phased Out

The following compilers are supported in Version 7.8 (R2009a), but will not be supported in a future version of MATLAB:

Windows (32-bit) Platforms

- Intel Visual Fortran Version 9.1
- Intel Visual Fortran Version 10.1
- Intel C/C++ Version 9.1
- Microsoft Visual Studio .NET Version 7.1

Windows (64-bit) Platforms

- Intel Visual Fortran Version 9.1
- Intel Visual Fortran Version 10.1
- Intel C/C++ Version 9.1

Solaris SPARC (64-bit) Platforms

- Sun Studio 11 cc / CC Version 5.8
- Sun Studio 11 f90 Version 8.2

Discontinued Compiler Support

MATLAB no longer supports the following compilers:

Windows (32-bit) Platforms

- Open Watcom Version 1.3
- Microsoft Visual Studio 2005 Express Edition

Windows (64-bit) Platforms

- Microsoft Platform SDK

Apple Macintosh (32-bit) Platforms

- g95 Version 0.90

Compatibility Considerations

To ensure continued support for building your C/C++ programs, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Do Not Use mxFree to Destroy mxArrays

It is improper to call `mxFree` on an `mxArray`. Previously, to remedy misleading statements in older documentation, under limited circumstances, MATLAB issued a warning in code that made this error. MATLAB no longer issues the warning.

Compatibility Considerations

The correct function to use to release memory for an `mxArray` is `mxDestroyArray`. Calling `mxFree` on an `mxArray` could cause memory corruption, which might result in a segmentation violation.

Cannot Build MEX-Files Using MATLAB Version 5 API

MATLAB does not support the `-V5` option to the `mex` function.

Compatibility Considerations

You are no longer able to build a MEX-file using the MATLAB Version 5 API. If you use any of the functions shown in the “Obsolete Functions: MX Array Manipulation” on page 33-24 table, you must

replace them with functions from the Replacement column, if available. These obsolete functions were deprecated when MATLAB Version 6 was released over 5 years ago.

MEX-Files Calling BLAS or LAPACK Functions Must Be Updated On 64-Bit Platforms

You must update any MEX-file that calls functions in the BLAS or LAPACK math packages on 64-bit platforms. The change occurs as a result of updated support, described in “64-bit Support in LAPACK and BLAS” on page 27-11. Existing MEX-files generated in previous versions of MATLAB will result in undefined behavior (likely crashes), if run in R2009a. The previous versions pass 32-bit integer arguments, but the math routines now read and write to 64 bits of memory. The results you see depend on what is stored in the subsequent 32 bits of memory.

Compatibility Considerations

On 64-bit platforms, you must use 64-bit integers for all input and output variables when calling LAPACK and BLAS routines in C and Fortran source MEX-files. Use the `mwSignedIndex` type for platform-independent code.

Object .o Files Saved on Macintosh Systems for Debugging

MATLAB saves object `.o` files when compiling MEX-files on Apple Mac OS Version 10.5 systems so that you can use source-level debugging.

Run-Time Libraries Required for Applications Built with Microsoft Visual Studio 2008 Compiler

If you distribute a MEX-file, an engine application, or a MAT-file application built with the Visual Studio 2008 compiler, you must provide the Visual C++ run-time libraries. These files are required to run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed. For information on locating the Microsoft Visual C++ 2008 Redistributable Package (x86), containing `vcredist_x86.exe` and `vcredist_x64.exe`, consult your Microsoft documentation.

New Features for Shared Library Interface

- It is now possible to use the `char**` return value and to increment the resulting pointer to retrieve all values. See Passing an Array of Strings.
- Added support for accessing values exported by a library.
- All fully and partly sized arrays should now work.

New Java Thread Safety Functions

Use the following new functions to work with Sun Java objects on the Event Dispatch Thread (EDT).

Function	Description
`javaObjectEDT`	Construct Java object on EDT
`javaMethodEDT`	Call Java method from EDT

Improved Robustness of Web Services Functions

The underlying technology used in the `createClassFromWsdl` and `parseSoapResponse` functions was modified to better ensure support for WSDL and SOAP standards.

Compatibility Considerations

There was no intended change to functionality or results of the `createClassFromWsdl` and `parseSoapResponse` functions, which MathWorks verified through testing. There are many variations among WSDL files and Web services and they cannot all be tested. Therefore, it is possible that your results using the `createClassFromWsdl` and `parseSoapResponse` functions in this version could differ from a previous version.

Ensure that your results using `createClassFromWsdl` and `parseSoapResponse` functions are as expected.

R2008b

Desktop Tools and Development Environment

Desktop New Features Video

For an overview of the major new features in the MATLAB Desktop Tools and Development Environment area, watch this video demo. Another way to access this and other video demos is to select the **Demos** tab in the Help browser, and then select **MATLAB > New Features in Version 7.7**.

Startup and Shutdown

New features and changes introduced in Version 7.7 (R2008b) are:

Macintosh Startup and Root Directory Enhancements and Changes

MATLAB for Apple Macintosh platforms is now installed like many other applications for Macintosh platforms, as a Macintosh `.app` bundle. This has resulted in some enhancements and changes to starting and using MATLAB on Macintosh platforms.

Startup Location

To start MATLAB on the Macintosh platform, double-click the `MATLAB_R2008b` icon in the `Applications` folder. This differs from R2008a (V7.6), where you started MATLAB by double-clicking the `MATLAB 7.6` icon in the `Applications/MATLAB_R2008a` folder; there was an additional folder level in R2008a.

Starting MATLAB from a File

You can now start MATLAB by double-clicking a file with a file extension that has been associated with MATLAB, such as a file with a `.m` extension. Similarly, you can drag a file onto the `MATLAB_R2008b` icon in the `Applications` folder or in the dock. These actions start MATLAB and open the file.

Contents of MATLAB Root Directory

When you use file browser GUIs to navigate in the MATLAB root directory, `/Applications/MATLAB_R2008b`, (known as the *`matlabroot`* directory), you cannot directly view or access its contents. For example, when you select **File** > **Open** and navigate to `Applications/MATLAB_R2008b`, no contents display. To access the contents, press **Command+Shift+G**, and enter the path to the MATLAB root directory in the resulting Go To Folder dialog box, for example, `/Applications/MATLAB_R2008b.app`.

Similarly, when you select `MATLAB_R2008b` in the `Applications` folder using the Finder, you do not see the contents. To access the contents, right-click (or **Ctrl**+click) `MATLAB_R2008b`, and from the context menu, select **Show Package Contents**. For more information, see Navigating Within the MATLAB Root Folder on Macintosh Platforms.

Startup Options and the Start MATLAB Settings Dialog Box

You can no longer set startup options using the Start MATLAB Settings dialog box and you can no longer start MATLAB from any `.smat` files you saved using Start MATLAB Settings. The dialog box is now used only for diagnostics and only appears if MATLAB experiences a problem during startup.

To set the startup directory in MATLAB, use the `userpath` function. To instruct MATLAB to run a specified statement upon startup, start MATLAB using the `matlab` command from a shell, as you would to start MATLAB on any UNIX platform. For more information, see Startup Options.

Starting MATLAB in a Shell

In R2008b (V7.7), to start MATLAB from a shell, enter the path to the executable, `/Applications/MATLAB_R2008b.app/bin/matlab`. This differs from R2008a (V7.6), in which the path was `/Applications/MATLAB_R2008a/bin/matlab`.

Compatibility Considerations

The compatibility considerations are described along with the above changes.

Updated Version of JVM Software

MATLAB is now using Sun Microsystems™ JVM™ Version 6 Update 4 software on all platforms, except the Apple Macintosh platform. If you specify a version of Java software to use with MATLAB, this change might impact your work.

Specifying Address Space Protection During Startup on Windows Platforms

When you start MATLAB on Microsoft Windows 32-bit platforms, you can set a startup option to help ensure the largest available contiguous block of memory after startup, which is useful if you run memory-intensive operations, such as processing large data sets. You can use the new `-shield` startup option to specify different levels of protection of the address space during the MATLAB startup process. This can also help resolve problems if MATLAB fails to start. For more information, see the `matlab (Windows)` reference page.

Changes to -nojvm Startup Option

When you start MATLAB with the `-nojvm` startup option, Handle Graphics functionality will no longer be supported. (The `-nojvm` option is available for UNIX platforms; see background information at the `matlab (UNIX)` reference page.)

Some aspects of MATLAB and related products use Handle Graphics in a way that might not be obvious. This includes anything that is based on or works using figures in MATLAB. Here is a summary of the affected features:

- Creating figures and performing plotting tasks, such as using the `plot`, `axes`, `getframe`, and `gcf` functions.
- Printing figures and using related functions such as `print`, `hgexport`, and `saveas`.
- Creating GUIs in MATLAB using GUI-building functions such as `warndlg`.
- Using Simulink scopes and printing Simulink models.

In MATLAB Version 7.7 (R2008b), if you use the `-nojvm` startup option and use Handle Graphics functionality, MATLAB produces this warning:

```
This functionality is no longer supported under the -nojvm startup
option.
```

In a future release, instead of a warning, MATLAB will produce an error when you start with the `-nojvm` option and use Handle Graphics functionality.

Compatibility Considerations

To avoid the warning, start MATLAB without the `-nojvm` startup option:

- If you have been using the `-nojvm` startup option to work in a command line environment or because you do not use the MATLAB desktop, use the `-nodesktop` startup option instead.
- If you have been using the `-nojvm` startup option because of memory or performance benefits, look for other ways to gain those improvements when you start MATLAB without the `-nojvm` option. See the Performance and Memory Usage topics in the MATLAB Programming Fundamentals documentation.
- If you want to continue to use the `-nojvm` startup option, remove the code that is now producing the warnings.

Changes to matlab Memory Manager Startup Options

The `matlab` command line arguments `-memmgr` and `-check_malloc` are deprecated and will be removed in a future release. The environment variable `MATLAB_MEM_MGR` is also deprecated and will be removed. For information about these options, see `matlab (Windows)` or `matlab (UNIX)`.

Compatibility Considerations

If you use these options, MATLAB generates a warning message.

Desktop

New features and changes introduced in Version 7.7 (R2008b) are:

New Default Layout for Desktop

There is a new desktop layout when you select **Desktop > Desktop Layout > Default**. It includes the same components as the previous default desktop layout, however, they are arranged differently. If you prefer the previous default layout, arrange your desktop in that way and save the layout. You then can reuse the saved layout at any time.

Closing Document Windows Using Middle Mouse Button

If you have multiple documents open, you can now close a document by clicking the middle mouse button when the pointer is in the document's button on the document bar.

Preferences Opens to Last Pane Used

When you open the Preferences dialog box, it displays the last preference pane you viewed in the current session. In prior versions, the Preferences dialog box displayed the pane associated with the tool from which you accessed it.

Changes to Desktop Text Font

You now can specify that the desktop text font use the system default font. To do this, select **File > Preferences > Fonts**. Then, for **Desktop text font**, select **Use system font**. For more information, click the **Help** button in the Fonts Preferences dialog box.

The default settings for the desktop text font and the HTML Proportional Text font have changed. This only affects existing users who choose to use a new preferences file (`matlab.prf`) in R2008b.

Provide Authentication Settings for Proxy Server when Accessing the Internet from MATLAB

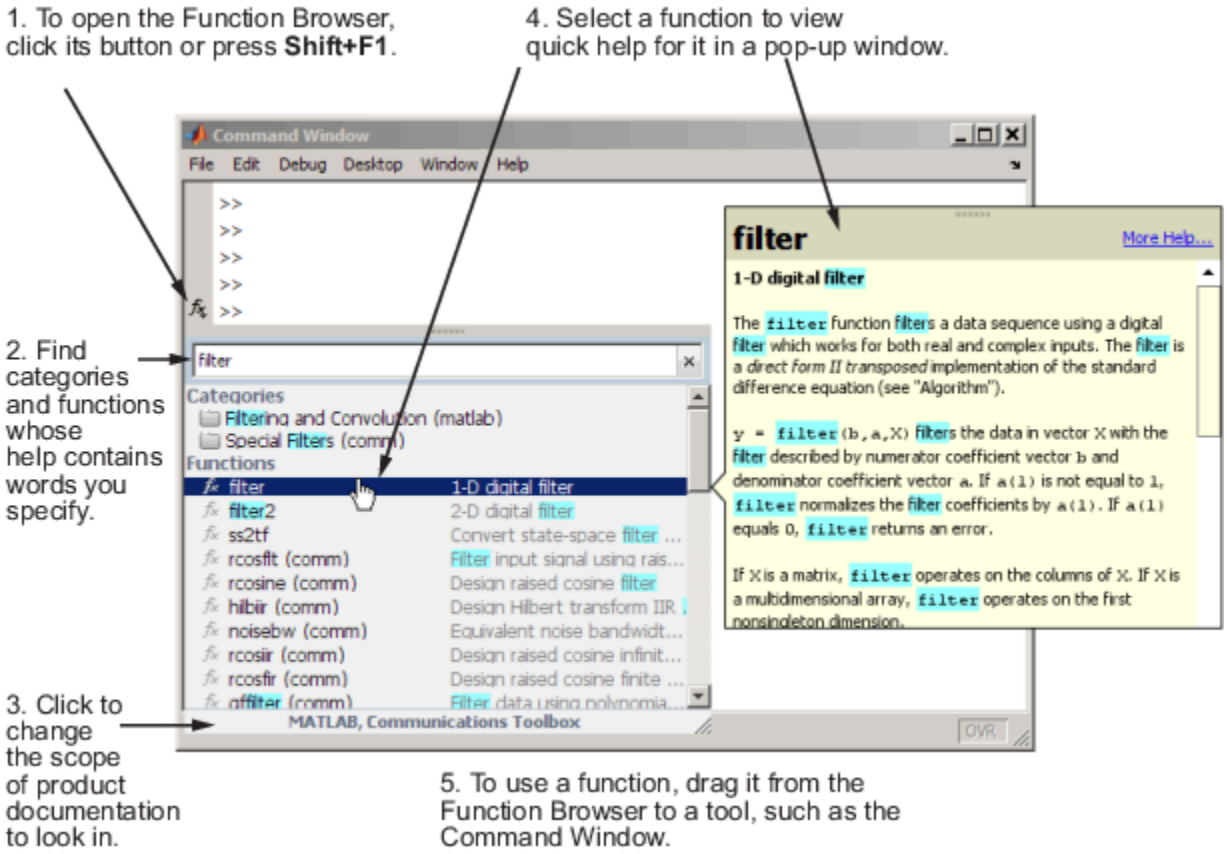
If you want to access the Internet from MATLAB and your network uses a firewall or another means of protection that restricts Internet access and requires you to provide a user name and password, use the new proxy server authentication settings in Web preferences to specify the values. For more information, click the **Help** button in the Web preferences pane, or see Web Preferences.

Running Functions — Command Window and History

Find Function Names and Get Help Using the New Function Browser

While you work, you can find the names of functions and get help for them using the new Function Browser. The Function Browser is useful if you want to find a function whose name you cannot remember, determine if there are functions that do what you want, or view the reference page for a function. The Function Browser uses a subset of the information found in the function reference pages in the Help browser for quick access while you work.

To access the Function Browser, press **Shift+F1**, or if you are in the Command Window or Editor, click the Function Browser button  . The Function Browser opens. You can specify the products to look in, browse categories of functions, and search for words that appear in function reference pages. For more information, see Find Functions Using the Function Browser.

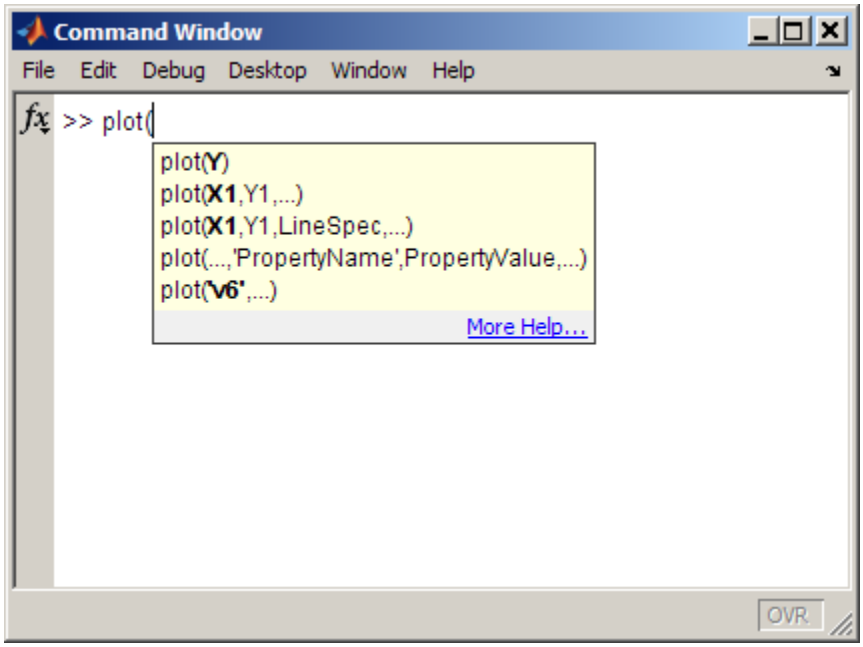


View Syntax Hints While Entering Statements

While you enter statements in the Command Window and Editor, you can view the allowable input argument syntax for a function in a pop-up window. This feature is called function hints. To use function hints:

1. Type the function name, followed by the left parenthesis, `(`, and pause. The syntax for input arguments automatically displays in a pop-up window near the statement you are entering.
2. The arguments you can enter appear in bold. Enter your first argument and type a comma (`,`) after it. The syntax options in the pop-up window then change, based on the argument you just entered.
3. Continue entering arguments, using the hints as needed. You can dismiss the function hints pop-up window at any time by pressing **Esc**. When you type the closing parenthesis, `)`, or when there are no more arguments to enter, the pop-up window automatically closes.

The following illustration shows function hints for the `plot` function.



To turn off the function hints feature so the pop-up menu does *not* automatically display the syntax reminders when you type an opening parenthesis, use Keyboard preferences; for **Function Hints**, clear the check box that enables them.

For more information, see View Function Syntax Hints While Entering a Statement.

Help and Related Resources

- “New Help Features — Function Browser and Function Hints” on page 28-7
- “Viewing an HTML Version of Help for Classes You Create” on page 28-7
- “Finding Text In Small Help Windows” on page 28-8
- “Changes to Search Field in Help Browser” on page 28-9

New Help Features — Function Browser and Function Hints

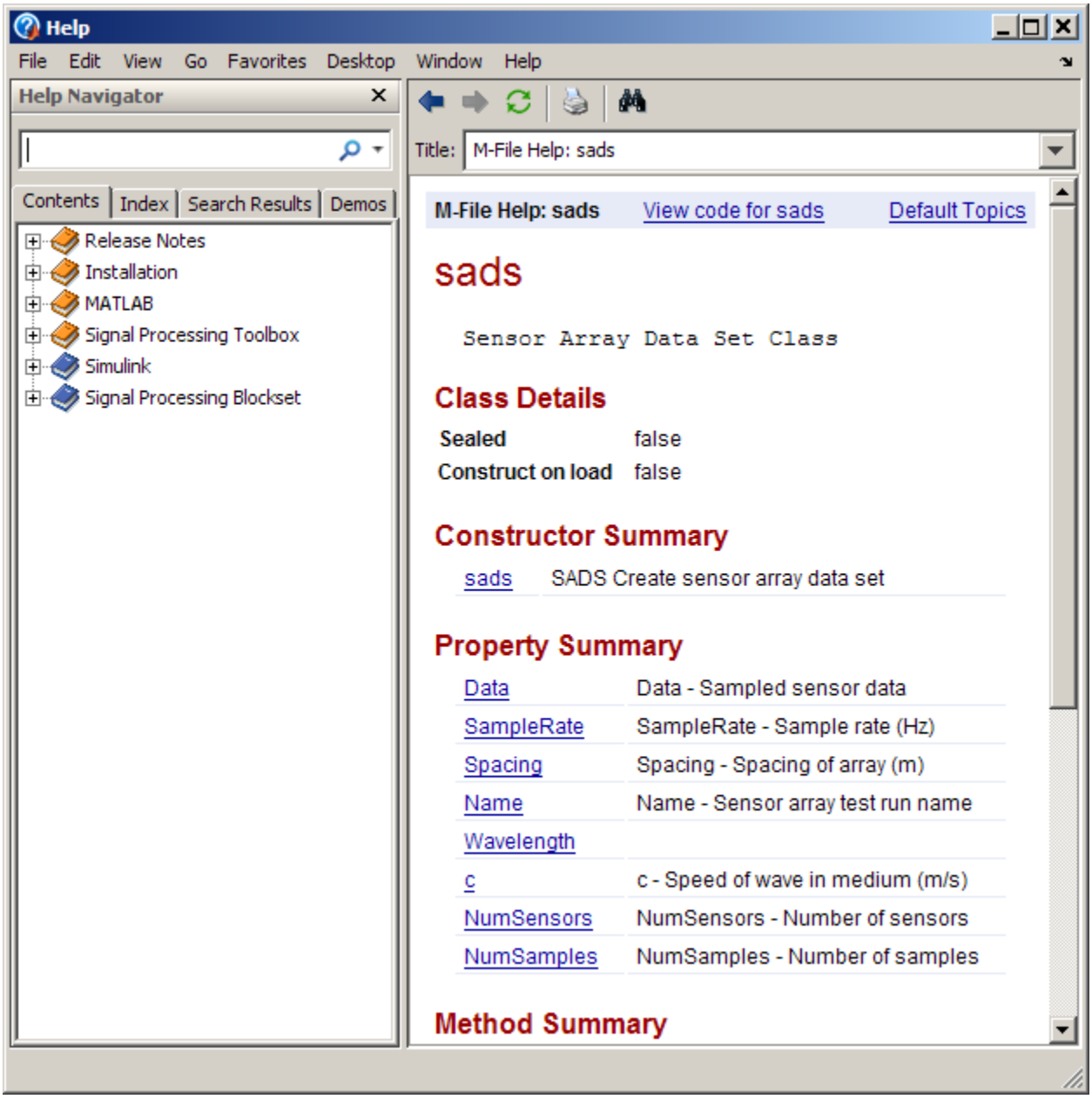
There are two new features that provide help while you work:

- Use the new Function Browser to find function names while you work. It is most useful in the Command Window and Editor, but you can access it from any tool. For more information, see “Find Function Names and Get Help Using the New Function Browser” on page 28-5.
- Use function hints to help you complete syntax for statements in the Command Window or Editor. For more information, see “View Syntax Hints While Entering Statements” on page 28-6.

Viewing an HTML Version of Help for Classes You Create

If you create your own class definition files in MATLAB and you provide help in the `classdef` file for the class, properties, and methods, you can conveniently view the help in the Help browser by running `doc` *`classname`*. For more information, see Help for Classes You Create. The following example shows class information in the Help browser for the user-created class, `sads`, displayed by running

```
doc sads
```



Finding Text In Small Help Windows

The Find dialog box is now available from the small help windows used for the Help on Selection feature and for context-sensitive help. The find feature is useful when you want to search for a specific word or phrase within one of these help windows. To access the Find dialog box:

- In a window used for the Help on Selection feature, press **Ctrl+F** on Windows and UNIX platforms or **Cmd+F** on Macintosh platforms. You can also click the **Find text** button
- In a context-sensitive help window, press **Ctrl+F** on Windows and UNIX platforms, and **Cmd+F** on Macintosh platforms

Changes to Search Field in Help Browser

You can quickly access the Product Filter from the search field by selecting the down arrow at the right side of the search field and from it, selecting **Filter by Product**. The Help pane of the Preferences dialog box opens. For more information about the product filter, click the **Help** button in the dialog box.

As you enter a term in the search field, a history of terms you previously entered in the current session appears. To view the full history, select the down arrow at the right side of the search field and from it, select **Show Search History**. You can select an item in the search history to rerun the search.

To execute a search, enter the search terms and press **Enter**. There is no longer a Go button.

Workspace, Search Path, and File Operations

New features and changes introduced in Version 7.7 (R2008b) are:

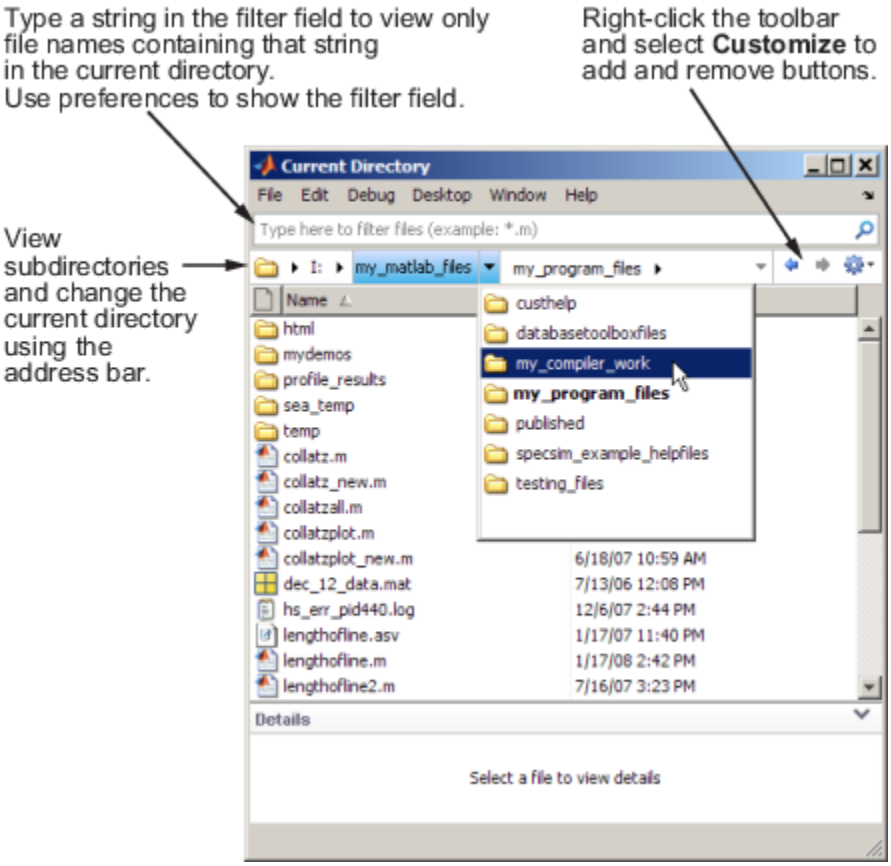
- "Current Directory Browser Enhanced, Including New Navigation and Grouping Features" on page 28-9
- "Structure Results of dir for Nonexistent Files Now Include Empty Matrices" on page 28-12
- "Workspace Browser Toolbar Is Now Configurable" on page 28-13
- "Semicolon (;) Path Separator Character Now Used on UNIX Platforms" on page 28-13

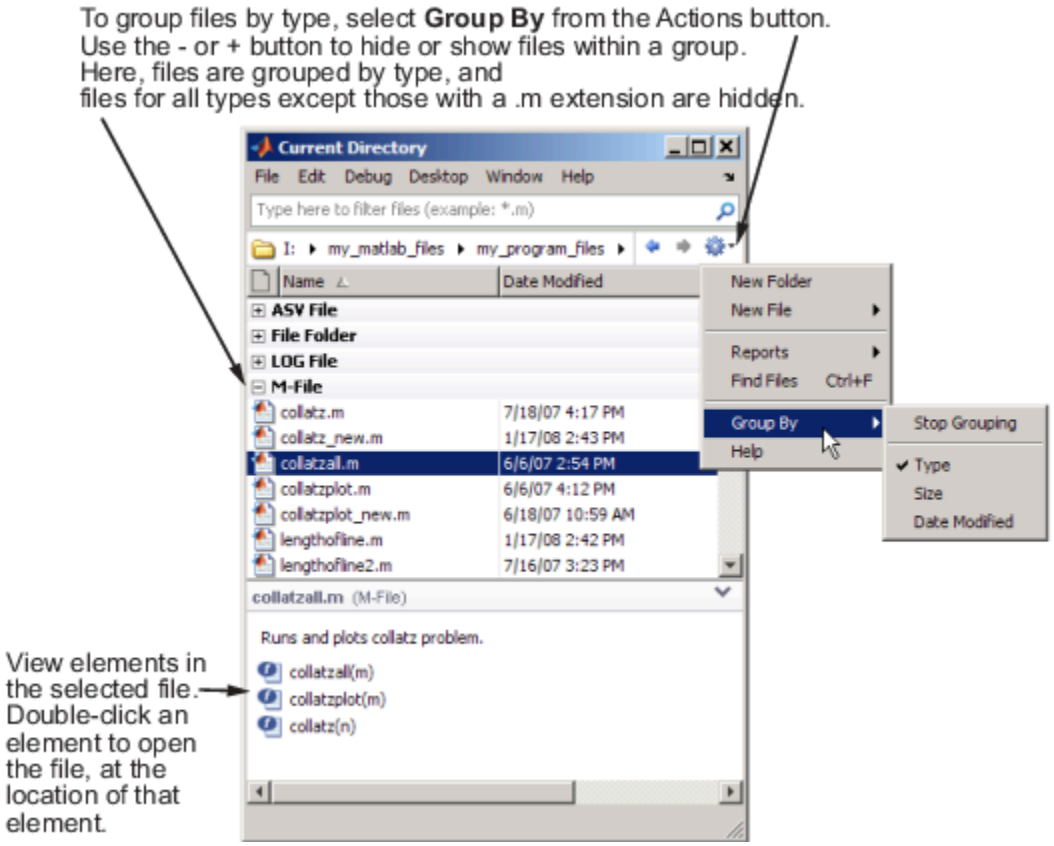
Current Directory Browser Enhanced, Including New Navigation and Grouping Features

The Current Directory browser includes new ways to navigate and to view the directory contents. There are also new ways to find files, including a new filter field.

- Use the new address bar to view the current directory and to view and navigate to subdirectories within the current directory path.
- Access common features from the Actions button  on the toolbar.
- Add and remove buttons from the toolbar by right-clicking the toolbar and selecting **Customize**.
- List only files whose names contain a specified string by using the new filter field. To show the filter field, use **File > Preferences > Current Directory**.
- In the **Details** pane at the bottom of the tool, view a list of elements in the selected file, such as subfunctions in a MATLAB program file. Double-click an element to open the file at the location of the element.
- For a MAT-file, drag selected variables from the **Details** pane to the Workspace browser to load them into MATLAB. Similarly, save workspace variables to a MAT-file by dragging them from the Workspace browser to the Current Directory browser.
- Group items in the current directory, that is, view related items together. From the Actions button, select **Group By** and select the attribute you want to group by.

Some of the major changes are highlighted in the following illustrations.


Type a string in the filter field to view only file names containing that string in the current directory.
Use preferences to show the filter field.
Right-click the toolbar and select **Customize** to add and remove buttons.
View subdirectories and change the current directory using the address bar.
Current Directory
File Edit Debug Desktop Window Help
Type here to filter files (example: *.m)
I: my_matlab_files my_program_files
Name
html
mydemos
profile_results
sea_temp
temp
collatz.m
collatz_new.m
collatzall.m
collatzplot.m
collatzplot_new.m
dec_12_data.mat
hs_err_pid440.log
lengthofline.asv
lengthofline.m
lengthofline2.m
custhelp
databasetoolboxfiles
my_compiler_work
my_program_files
published
specsim_example_helpfiles
testing_files
6/18/07 10:59 AM
7/13/06 12:08 PM
12/6/07 2:44 PM
1/17/07 11:40 PM
1/17/08 2:42 PM
7/16/07 3:23 PM
Details
Select a file to view details



For more information, see Managing Files in MATLAB.

Compatibility Considerations

The following aspects of using the Current Directory browser are different in Version 7.7 (R2008b) than in the previous version. Following is a table of what changed and the way to perform the same action in Version 7.7.

Change	What To Do Instead In Version 7.7 (R2008b)
The toolbar no longer includes the buttons it previously included.	You can now specify which buttons to include on the toolbar. You can include the toolbar buttons included in previous versions, or use these alternatives: • Go up one level button — Instead, select a directory one level up by using the address bar. • Find files button — Instead, select **Find Files** from the Actions button on the toolbar, or use **Ctrl+F**. • New folder button — Instead, select **New Folder** from the Actions button on the toolbar. • Directory reports — Instead, select **Reports** from the Actions button on the toolbar.

Change	What To Do Instead In Version 7.7 (R2008b)
The string display of the full path for the current directory has been replaced by the address bar view.	You can still access the string display, which can be useful if you want to copy it or edit it directly. To access the string, click the blank area to the right of the last subdirectory in the address bar. To return to the address bar view, press the **Escape** key.
You do not select attributes (columns) to display using Current Directory Preferences (**Browser display options**).	Right-click any column header to select or clear the columns to display, such as file size. You can only do this on Microsoft Windows platforms.
The file description no longer appears in a column.	View the description for a selected file in the detail pane.
The full help for a MATLAB program file no longer appears in the detail pane.	Press **F1** to view the reference page for the selected file.
You do not elect to show details using Current Directory Preferences (**Browser display options**).	Use the arrow on the right side of the detail pane separator to show or hide the details.
The **File Filter** options, accessible from the context menu and the **View** menu, have been removed.	Use the new filter field (use Preferences for the Current Directory browser to show the filter field), or sort by file type (click the document icon header), or group by file type using the new grouping feature (available from the Actions button).
The **View** menu was removed.	Use the new filter field (previously described) to filter the view. For Directory Reports, select **Reports** from the Actions button on the toolbar.
When you look for items in the current directory by typing the initial letters in the name, a pop-up window no longer appears showing the letters that you typed.	As before, the first entry whose name begins with the letters you typed is selected. If you want to see what you type when finding items in the current directory, instead use the filter field.

Structure Results of dir for Nonexistent Files Now Include Empty Matrices

When you run `dir` with an output argument and the results include a nonexistent file or a file that `dir` cannot query for some other reason, `dir` now returns empty matrices for the `date`, `bytes`, and `datenum` fields. The most common occurrence is on UNIX platforms when `dir` queries a file that is a symbolic link and the symbolic link points to a nonexistent target. A nonexistent target is when a target has been moved, removed, or renamed. For example, if `my_file` in `my_dir` is a symbolic link to another file that has been deleted, then running

```
r = dir('my_dir')
```

includes this result for `my_file`:

```
r(n) =
        name: 'my_file'
        date: ''
        bytes: []
        isdir: 0
        datenum: []
```

where *n* is the index for `my_file`, found by searching `r` by the `name` field.

With empty matrices returned, code you write to use the results of `dir` can be simpler and more robust. For more information, see the reference page for the `dir` function.

Compatibility Considerations

In previous versions, `dir` returned inappropriate values for files that could not be queried:

```
r(n) =
    name: 'my_file'
    date: '18-Jan-2038 22:14:07'
    bytes: 0
    isdir: 0
    datenum: 7.4438e+05
```

If you have existing files that rely on the results of `dir` for a file that could not be queried, your code might not produce the result it used to. If you write new code that depends on the new results for `dir` and run it in a previous version of MATLAB, it might not produce the results you expect. In either case, there will probably not be a warning or error.

Workspace Browser Toolbar Is Now Configurable

Rearrange, add, or remove buttons and other controls from the Workspace Browser toolbar using **File** > **Preferences** > **Toolbars**. Alternatively, right-click the toolbar and select **Customize** from the context menu. By default, the Print button is not on the toolbar.

For more information on customizing the toolbar, see Toolbar Customization.

Semicolon (;) Path Separator Character Now Used on UNIX Platforms

On UNIX platforms, MATLAB now uses a semicolon (;) as the character that separates directories in the search path file, `pathdef.m`, which is the character used on Windows platforms. This is beneficial if you work with the contents of the `pathdef.m` file programmatically on both Windows and UNIX platforms. In versions prior to Version 7.7 (R2008b), MATLAB used the colon (:) character as the path separator on UNIX platforms.

Compatibility Considerations

MATLAB will continue to accept the colon (:) character as a valid path separator on UNIX platforms, so any existing code you have that relies on it will still work.

You will experience a problem if you have any directories that contain a semicolon in their name. You will need to rename the directories to include them on the search path.

Editing and Debugging MATLAB Code

New features and changes introduced in Version 7.7 (R2008b) are:

- “Create New Function and Class Files Using Templates” on page 28-14
- “View Syntax Hints, Find Function Names, and Get Quick Help” on page 28-14
- “Set Color and Width of Right-Hand Text Limit” on page 28-14

Create New Function and Class Files Using Templates

When you create new functions and class definition files (`classdef`), you can start with a template that reminds you to include standard information. Select **File > New > Function M-File**, or **File > New > Class M-File**. A new file containing template information opens in the Editor.

View Syntax Hints, Find Function Names, and Get Quick Help

While you enter statements in the Editor or Command Window, you can display the syntax for a function in a temporary pop–up window. For more information, see “View Syntax Hints While Entering Statements” on page 28-6.

You also can find the names of and get help for functions using the new Function Browser—for more information, see “Find Function Names and Get Help Using the New Function Browser” on page 28-5.

Set Color and Width of Right-Hand Text Limit

In the Editor, where you can enable a vertical line to indicate a right-hand text limit, you now can set the width and color of the line. Note that the default color for the line is now gray, instead of light red.

Set preferences for the line by selecting **File > Preferences > Editor/Debugger > Display**, which opens the Preferences dialog box.

Click **Help** in the Preferences dialog box for more information.

Set Cursor to First Nonwhite Character on Line

When you press the **Home** key, the cursor goes to the first nonwhite character on the current line. When you press the **Home** key twice, the cursor goes to the beginning of the line.

Compatibility Considerations

In versions prior to Version 7.7 (R2008b), the **Home** key always moved the cursor to the first column of the current line.

Suppress a Specific M-Lint Message Throughout a File

You can now suppress a specific M-Lint message throughout a file by right-clicking on an M-Lint indicator that elicits the message, and then from the context menu, selecting `Suppress All Instances in this File`.

For details, see Suppress All Instances of a Message in the Current File in the MATLAB Desktop Tools and Development Environment documentation. For information on manually inserting the string that tells M-Lint to suppress the message throughout the file, see the documentation for the `mlint` function.

New M-Lint Message for Suppressed Messages

An M-Lint message now appears if you previously suppressed a message using the `%#ok` directive, and that message no longer appears. This can result if the message is already suppressed using preferences (**File > Preferences > M-Lint**), and the code has changed such that the message is no longer generated, or if the rules M-Lint follows for generating the message have changed. The new message is:

```
An M-Lint message was suppressed here, but the message no longer
appears. Use the context menu to fix.
```

For an example, see the *Suppressing All Messages on a Line with mlint* example in the `mlint` function documentation.

View M-Lint Message in ToolTip Using the Keyboard

To open an M-Lint Message ToolTip using the keyboard, place the cursor over the marked code and press **Ctrl + M**. This feature is offered in addition to the identical behavior available when you use the mouse pointer to hover over code that is marked by M-Lint. For an example of viewing an M-Lint message in a ToolTip, see Check Code for Errors and Warnings in the MATLAB Desktop Tools and Development Environment documentation.

Apply M-Lint Autofix Using the Keyboard

To fix a problem marked by M-Lint as having an automatic fix available, place the cursor over the marked code, and then press **Alt + Enter**. This feature is offered in addition to the identical behavior available when you use the context menu. For an example of using the M-Lint autofix feature, see Check Code for Errors and Warnings in the MATLAB Desktop Tools and Development Environment documentation.

Code Fold Single Program, Multiple Data (spmd) Blocks

By default, the Editor now supports code folding for single program, multiple data (spmd) blocks. For more information on code folding, see Code Folding — Expand and Collapse Code Constructs in the MATLAB Desktop Tools and Development Environment documentation.

File and Directory Comparisons Tool: Highlight Changes

The File and Directory Comparisons Tool now uses shades of colors to mark differences in the contents of two directories being compared. Light colors indicate files that differ and dark colors indicate subdirectories that differ. For details and an example, see Comparing Folders and Zip Files in the MATLAB Desktop Tools and Development Environment documentation.

Compatibility Considerations

In releases prior to Version 7.7 (R2008b), the same color intensity highlighted differences in both files and directories.

Block Indenting Will Not Be Included in Next Version

The **Block Indent** option will no longer be provided, starting in the next version of MATLAB. Currently, this option is available for M, Java, and C/C++ programming languages, when you select **File > Preferences > Editor/Debugger > Language**. To attain the effect of block indenting, you can use the **No indent** option and indent lines manually using the **Tab** and **space** keys.

If you have concerns about the pending removal of the **Block Indent** option, please contact Technical Support at `https://www.mathworks.com/support/contact_us`.

Accessing Contents of MATLAB Root Directory on Macintosh Platforms

Starting in MATLAB 7.7 (R2008b), on Macintosh platforms, you cannot use file browser GUIs to directly access contents of the MATLAB root directory. For example, when you use **File > Open** from the Editor, you cannot directly select a file located within *`matlabroot`*. For more information, see “Contents of MATLAB Root Directory” on page 28-2.

Tuning and Managing MATLAB Code Files

Access Directory Reports

You now access Directory Reports by navigating to the directory containing the MATLAB program files for which you want to produce reports. Then, on the Current Directory browser toolbar, click the **Actions** down arrow  and select the type of report you want to run for all of the MATLAB program files in the current directory. Note that these reports are now referred to as Reports, rather than Directory Reports.

Compatibility Considerations

In versions prior to Version 7.7 (R2008b), you navigated to the directory containing the MATLAB program files for which you wanted to produce reports. Then, you clicked the **Directory Reports** down arrow  on the Current Directory browser toolbar.

Publishing MATLAB Code Files

New features and changes introduced in Version 7.7 (R2008b) are:

- “Include Figure Window Details in Published Documents” on page 28-16
- “Inline Math Supported in Published Documents” on page 28-17
- “Publish Setting: Cascading Style Sheet Is Now XSL File” on page 28-17

Include Figure Window Details in Published Documents

In versions prior to Version 7.7 (R2008b), when you published a file that included a figure window, only the graph or figure was included in the published document. Using the `figureSnapMethod` option, you can now specify that you want the window details included in the published document. For details, see the `publish` reference page.

Inline Math Supported in Published Documents

In versions prior to Version 7.7 (R2008b), you could publish LaTeX code in a published document as a code block, separate from the rest of your comments, if any. Now, you can publish LaTeX math symbols inline with the rest of your comments. For details, see Inline LaTeX Math Equations.

Publish Setting: Cascading Style Sheet Is Now XSL File

In versions prior to Version 7.7 (R2008b), there was a **Cascading Style Sheet** setting on the Publish Configurations dialog box (which you access by clicking the Publish down-arrow button ). This setting is now called **XSL File**, to more accurately reflect the setting.

Compatibility Considerations

The **Cascading Style Sheet** setting on the Publish Configurations dialog box is now **XSL File**.

Mathematics

Upgrade to Random Number Generator

- The `randn` function uses a new longer period random number algorithm as its default.
- The new function `randi` returns random integers from a uniform discrete distribution.
- The `RandStream` class allows you to construct a random number stream object and set its properties. For more information, see Random Numbers in the MATLAB Mathematics documentation.

Compatibility Considerations

The `randn` function now produces different results than in previous releases. Because the values returned by `randn` are intended to be random, this change should not affect most code.

`rand` and `randn` now draw from the same random number stream. In prior releases, `rand` and `randn` had separate independent underlying random number streams. Since `rand` and `randn` now access the same stream, using `randn` will affect subsequent values produced by `rand` and vice-versa. See Creating and Controlling a Random Number Stream in the MATLAB Mathematics documentation for more information.

Multipoint Boundary-Value Problems with bvp5c

The solver `bvp5c` will take multipoint boundary-value problems.

Upgrades to lsqnonneg

`lsqnonneg` now runs more efficiently. It accepts sparse matrices as inputs and maintains sparsity throughout its internal iterations.

Functions and Properties Being Removed

Function or Property Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`betacore`	Fails	`betainc`	Replace all existing instances of `betacore` with `betainc`.
`colmmd`	Fails	`colamd`	Replace all existing instances of `colmmd` with `colamd`.
`symmmd`	Fails	`symamd`	Replace all existing instances of `symmmd` with `symamd`.
`flops`	Fails	None	Remove all instances of `flops`. With the incorporation of LAPACK in MATLAB Version 6, counting floating-point operations is no longer practical.

Upgrade to Intel Math Kernel Libraries

For Windows, Intel Mac, and Linux platforms, MATLAB software supports the Intel Math Kernel Library (MKL) version 10.0.3.

Data Analysis

Specialized Data Tips for the hist Function

When you create a histogram display using `hist` and place data tips in the plot in data cursor mode, they now snap to the top center of the bin you attach them to. The data tip contents have changed as well, and now consist of:

- Number of observations falling into the selected bin
- The x-value of the bin's center
- The lower and upper x-values for the bin

For details, see Using Data Cursors with Histograms in the MATLAB Graphics documentation.

Compatibility Considerations

Data tips for histograms no longer display the *x* and *y* coordinates of their locations and no longer snap to the four corners of the bins, as they do for `bar` plots. The data tips are also larger, to accommodate the extra information.

Programming Fundamentals

Fast Key Lookup Provided with New Map Data Structure

This release introduces a new MATLAB class called a Map. An object of the Map class is an array of any MATLAB data type that supports lookup table functionality. Unlike most arrays in MATLAB that only allow access to the elements by means of integer indices, indices for Map containers can be nearly any scalar numeric value or a character string.

The following example creates a Map object that is an array of strings. It is very much like any other string array except that with each value of this array there is also a lookup key associated with it. This particular Map contains the names of capital cities in the United States. These are the values of the Map object. Associated with each capital city value is the US state that it resides in. These are the keys of the Map object. You look up values in the Map using key indices. Call the `containers.Map` constructor to create an array of six capital cities indexed by six US states. (The capital of Alaska has purposely been entered incorrectly):

```
US_Capitals = containers.Map( ...
{'Arizona', 'Nebraska', 'Nevada', ...       % 6 States
 'New York', 'Georgia', 'Alaska'}, ...
{'Phoenix', 'Lincoln', 'Carson City', ...   % 6 Capitals
 'Albany', 'Atlanta', 'Fairbanks'});
```

Show the capitals of three of the states by looking them up in the Map using the string indices `'Nevada'`, `'Alaska'` and `'Georgia'`:

```
values(US_Capitals, {'Nevada', 'Alaska', 'Georgia'})
ans =
    'Carson City'    'Fairbanks'    'Atlanta'
```

Correct the capital city of Alaska by overwriting the entry at string index `'Alaska'`:

```
US_Capitals('Alaska') = 'Juneau';

US_Capitals('Alaska')
ans =
   Juneau
```

The term `containers.Map` refers to a `Map` class that is part of a MATLAB package called `containers`. For more information, see Map Containers in the Programming Fundamentals documentation.

Tic and Toc Support Multiple Consecutive Timings

The `tic` and `toc` timing functions now support multiple consecutive timings. Call `tic` with an output `t0` to save the current time as the starting time for some operation. When the operation completes, call `toc` with the same `t0` as its input and MATLAB displays the time between that particular tic and toc.

In the following example, MATLAB measures the time used by each function call and, at the same time, measures the time required for the overall operation:

```
t0 = tic;
   t1 = tic;   W = myfun1(A,B);        toc(t1)
```

```
   t2 = tic;   [X,Y] = myfun2(C,W);    toc(t2)
   t3 = tic;   Z = myfun3(A,C,Y);      toc(t3)
toc(t0)
```

You can still call `tic` and `toc` without any arguments. In this case, `toc` just measures the time since the most recent `tic`.

See the function reference page for `tic` or `toc` for more information.

New Options for MException getReport

The MException `getReport` method has several new options available in this release. You select these options when you call `getReport`. They give you more control over the content and format of the information displayed or returned by `getReport`.

See the function reference page for `getReport` for more information.

what Function Returns Package Information

The `what` function now includes package information in its output display and a `package` field in the structure array that it returns.

List the packages used in the MathWorks Communications Toolbox™:

```
s = what('comm');

s.packages
ans =
    'crc'
    'commdevice'
    'commsrc'
    'commgui'
    'commscope'
    'commutils'
```

You can also call `what` on a specific package name to see what types of directories and files are in the package directory.

See the function reference page for `what` for more information.

addtodate Accepts Hours, Minutes, Seconds, Milliseconds

In previous releases, the `addtodate` function supported modifying a date number by a specified number of years, months, or days. In this release, you can also modify the date by a specified number of hours, minutes, seconds, or milliseconds.

Add 2 hours, 45 minutes, and 17 seconds to the current time:

```
d1 = now;
datestr(d1)
ans =
   12-Jun-2008 16:15:38

d2 = addtodate(d1, 2, 'hour');
d2 = addtodate(d2, 45, 'minute');
```

```
d2 = addtodate(d2, 17, 'second');
d2 = addtodate(d2, 3000, 'millisecond');

datestr(d2)
ans =
   12-Jun-2008 19:00:58
```

See the function reference page for `addtodate` for more information.

Querying Options Added to pause

There are new syntaxes for the `pause` function:

- To see whether pausing is enabled or not, use one of the following commands:

  ```
  pause query
  state = pause('query')
  ```

- To return the previous pause state when enabling or disabling pausing, use one of the following:

  ```
  oldstate = pause('on')
  oldstate = pause('off')
  ```

See the function reference page for `pause` for more information.

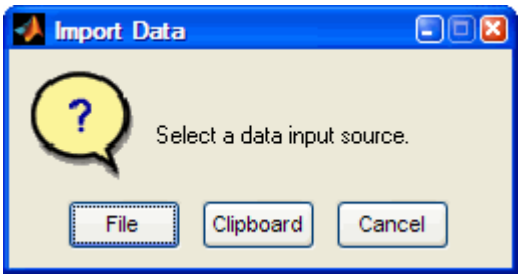
File Selection Restriction in Import Wizard

This release introduces a change in how you select the source to import using the Import Wizard. In previous releases, you could select and view the contents of any number of files within the wizard before choosing the one to import. As of this release, if you need to import from a different source (a specific file or the system clipboard) than the one you had originally selected, you must exit and restart the Import Wizard. This change gives the Import Wizard increased flexibility in handling different types of files, removes redundancy in the import process, and also removes a potential source of unexpected behavior from the product.

Compatibility Considerations

The user interface to the Import Wizard is very much the same as in previous versions of MATLAB. However, you should take note of the following changes:

- The panel named **Select Data Source**, that appears at the top of the Import Wizard preview dialog box in earlier releases, is no longer available. As before, you select the source file to import, or the clipboard, at the time you activate the Import Wizard. This applies whether you use **File** > **Import Data** from the MATLAB Command Window menu or the `uiimport` function at the command prompt. To make a new file or clipboard selection, click the **Cancel** button in any of the dialog boxes, and then restart the Wizard.
- Calling `uiimport` without a file name displays the following new dialog box. Choosing **File** opens the Import Data dialog box. Selecting **Clipboard** opens the wizard with the contents of the clipboard in the preview panel.



- To import from the clipboard using the MATLAB menus, use **Edit > Paste to Workspace** instead of using **File > Import Data** , and then change the source to **Clipboard**. This method is preferred because it is more direct. The capability of switching from file to clipboard within the wizard is no longer supported.

Function Handle Array Warning Is Now An Error

The only way to make an array of function handles is to use a cell array. Attempting to create any other type of array of function handles is invalid. For the past several releases, MATLAB has issued a warning if you should attempt to put function handles into any type of array other than a cell array. As of this release, MATLAB throws an error instead of a warning when this is attempted.

Replace

```
A = [@sin @cos @tan];
??? Error using ==> horzcat
Nonscalar arrays of function handles are not allowed; use
cell arrays instead.
```

with

```
A = {@sin @cos @tan};
```

Compatibility Considerations

If any of your program code attempts to create a regular array of function handles, this code will now generate an error.

Two Types of issorted Warnings Are Now Errors

In previous versions, the `issorted` function generated a warning for the following two cases:

- `issorted(x)`, where `x` is a complex integer, and
- `issorted(x,'rows')`, where `x` is an ND array

Using issorted on a Complex Integer

In this case, a statement such as the following

```
issorted(int8(complex(1,2)))
```

now issues the error message

```
??? Error using ==> issorted
ISSORTED on complex inputs with integer class is obsolete.
Please use ISSORTED(DOUBLE(X)) or ISSORTED(SINGLE(X)) instead.
```

Using issorted on an N-D Array

In this case, a statement such as

```
issorted(ones(3,3,3),'rows')
```

now issues the error message

```
??? Error using ==> issorted
X must be a 2-D matrix.
```

Compatibility Considerations

If any of your program code attempts to use either of these types of statements, this code will now generate an error.

Possible Conflict with New Keyword: SPMD

This release introduces a new keyword, `spmd`, that, although used solely by the Parallel Computing Toolbox (PCT), may cause conflicts with MATLAB users as well. See spmd Construct in the PCT release notes for more information on this keyword.

Compatibility Considerations

Because `spmd` is a new keyword, it will conflict with any user-defined functions or variables of the same name. If you have any code with functions or variables named `spmd`, you must rename them.

Do Not Create MEX-Files with DLL File Extensions

In the future, on 32-bit Windows systems, MATLAB will not support MEX-files with a `.dll` file extension. See release note “Do Not Use DLL File Extensions for MEX-Files” on page 28-31 for information on how this might affect you.

isequal Is Now Called Explicitly for Contained Objects

When you pass two MATLAB objects to `isequal`, MATLAB dispatches to the `isequal` method of the dominant object (see Object Precedence in Expressions Using Operators). If the dominant object does not overload `isequal`, then MATLAB uses the built-in version.

Compatibility Considerations

What is different with this release, is that MATLAB now calls `isequal` explicitly for each contained object. This means that, if any contained object overloads `isequal`, MATLAB calls the overloaded version for that object.

Previously, MATLAB compared contained objects using the built-in `isequal` functionality without regard to any special behavior programmed into overloaded `isequal` methods. The effect of this

change is that, for objects that contain other objects and those contained objects overload `isequal`, the overloaded behavior establishes the basis with which MATLAB determines equality for those contained objects.

Indexed Assignment with Objects of the Form p(:) = o Now Consistent with MATLAB Language

The behavior of indexed assignment with MATLAB objects is consistent with the behavior of all MATLAB intrinsic types and V5 MATLAB objects. For example, attempting the following assignment, where `d` does not previously exist, gives an error:

```
>> d(:) = 5;
???  In an assignment  A(:) = B, the number of elements in A and B
must be the same.
```

MATLAB objects behave in the same way:

```
ts_obj = timeseries;
t(:) = ts_obj;
???  In an assignment  A(:) = B, the number of elements in A and B
must be the same.
```

Compatibility Considerations

In MATLAB Version 7.6 Release 2008a, indexed assignment of the form `p(:) = object` did not result in an error.

fopen No Longer Supports VAXD, VAXG, and Cray Machine Formats

Calls to `fopen` with any of the following values for machine format return an error:

- `'vaxd'` or `'d'`
- `'vaxg'` or `'g'`
- `'cray'` or `'c'`

Compatibility Considerations

In previous releases, calls to `fopen` with machine format values associated with VAXD, VAXG, and Cray did not result in an error.

To read files in VAXD and VAXG formats, consider the workaround available on the MATLAB Central File Exchange, file ID #22675.

Graphics and 3-D Visualization

Certain Printer Formats and Drivers Now Warn When Used

Going forward, MathWorks is planning to leverage existing operating system (OS) support for printer drivers and devices. As a result, the ability to specify certain print devices using the `print -d` command, and certain graphics formats using the `print -d` command and/or the `saveas` command, will be removed in a future release.

Graphic Format Drivers		
`-pkm`	`-pkmraw`	`-tifflzw`

Printer Driver	**`print` Command Option String**
Canon® BubbleJet BJ10e	`-dbj10e`
Canon BubbleJet BJ200 color	`-dbj200`
Canon Color BubbleJet BJC-70/BJC-600/ BJC-4000	`-dbjc600`
Canon Color BubbleJet BJC-800	`-dbjc800`
Epson® and compatible 9- or 24-pin dot matrix print drivers	`-depson`
Epson and compatible 9-pin with interleaved lines (triple resolution)	`-deps9high`
Epson LQ-2550 and compatible; color (not supported on HP®-700)	`-depsonc`
Fujitsu® 3400/2400/1200	`-depsonc`
HP DesignJet 650C color (not supported on Windows OS)	`-ddnj650c`
HP DeskJet 500	`-ddjet500`
HP DeskJet 500C (creates black and white output)	`-dcdjmono`
HP DeskJet 500C (with 24 bit/pixel color and high-quality Floyd-Steinberg color dithering) (not supported on Windows OS)	`-dcdjcolor`
HP DeskJet 500C/540C color (not supported on Windows OS)	`-dcdj500`
HP DeskJet 550C color (not supported on Windows OS)	`-dcdj550`
HP DeskJet and **DeskJet Plus**	`-ddeskjet`
HP LaserJet	`-dlaserjet`
HP LaserJet+	`-dljetplus`
HP LaserJet IIP	`-dljet2p`
HP LaserJet III	`-dljet3`
HP LaserJet 4.5L and 5P	`-dljet4`

Printer Driver	`print` Command Option String
HP LaserJet 5 and 6	`-dpxlmono`
HP PaintJet color	`-dpaintjet`
HP PaintJet XL color	`-dpjxl`
HP PaintJet XL color	`-dpjetxl`
HP PaintJet XL300 color (not supported on Windows OS)	`-dpjxl300`
HP-GL® for HP 7475A and other compatible plotters. (Renderer cannot be set to Z-buffer.)	`-dhpgl`
IBM® 9-pin Proprinter	`-dibmpro`

Handle Graphics Not Supported Under -nojvm Startup Option

If you start MATLAB with `matlab -nojvm` (which disables Java) you will receive a warning when you attempt to create or load figures, open GUIs, print or capture figures using `getframe`.

Compatibility Considerations

For information, see “Changes to -nojvm Startup Option” on page 28-3 in the Desktop Tools and Development Environment release notes.

Creating Graphical User Interfaces (GUIs)

Undocumented Functions Removed

The following set of deprecated functions, all of which were previously undocumented, have been removed. Alternatives to most of them exist, which are described.

- `axlimdlg` — No alternative
- `cbedit` — No alternative
- `clruprop` — Use `rmappdata` instead
- `ctlpanel` — No alternative
- `edtext` — Set text object's `Editing` property
- `extent` — Get text object's `Extent` property
- `getuprop` — Use `getappdata` instead
- `hidegui` — Set figure's `HandleVisibility` property
- `hthelp` — Use `web` instead
- `layout` — No alternative
- `matq2ws` — Combine `save` + `load` or `uisave` + `uiload`
- `matqdlg` — Combine `save` + `load` or `uisave` + `uiload`
- `matqparse` — Combine `save` + `load` or `uisave` + `uiload`
- `matqueue` — Combine `save` + `load` or `uisave` + `uiload`
- `menubar` — The string `'none'`
- `menuedit` — No alternative
- `pagedlg` — Use `pagesetupdlg` instead
- `setuprop` — Use `setappdata` instead
- `umtoggle` — Set uimenu's `Checked` property
- `wizard` — Use `guide` instead
- `ws2matq` — Combine `save` + `load` or `uisave` + `uiload`

Compatibility Considerations

If you have developed MATLAB code that uses any of the above undocumented functions, you must remove such calls or replace them with other code, as suggested within the above bullets.

Handle Graphics Not Supported Under -nojvm Startup Option

If you start MATLAB with `matlab -nojvm` (which disables Java) you will receive a warning when you attempt to create or load figures, open GUIs, print, or capture figures using `getframe`. For more information, see “Changes to -nojvm Startup Option” on page 28-3 in the Desktop Tools and Development Environment release notes.

New Menu Options to Hide or Show GUIDE Toolbar and Status Bar

Two new menu options in GUIDE let you hide/show the GUIDE toolbar and status bar. By default both are visible, but you can deselect **Show Toolbar**, **Show Status Bar**, or both from the **View** menu to make the layout area larger.

Compatibility Considerations

The GUIDE Preferences option **Show Toolbar** is no longer available. Use **Show Toolbar** from the GUIDE **View** menu instead.

GUIDE Status Bar Now Shows Tag Property of Selected Object

The GUIDE Layout Editor now shows the `Tag` property of any object you select, in the left corner of the status bar that runs along its bottom. This display saves you from having to open the Property Inspector to read `Tag` names. Information in the status bar is read-only; you still need to use the Property Inspector to change a component's `Tag`.

Four New Major GUI Examples

The Creating Graphical User Interface documentation has four new extensive examples of building GUIs with GUIDE and programmatically. All of them feature uitables, a feature introduced on page 29-33 in R2008a, and all the GUIs plot data. The new examples are:

- A Working GUI with Many Components (GUIDE)
- GUI for Animating a 3-D View (GUIDE) (GUIDE)
- GUI to Interactively Explore Data in a Table (GUIDE) (GUIDE)
- GUI that Displays and Graphs Tabular Data (programmatic)

This release includes FIG- and code files for all these examples. The documentation discusses many of their callbacks and includes hyperlinks to code in the code files.

External Interfaces/API

Do Not Use DLL File Extensions for MEX-Files

In the future, on 32-bit Microsoft Windows systems, MATLAB will not support MEX-files with a `.dll` file extension. In MATLAB Version 7.7 (R2008b), if you run a MEX-file with a `.dll` file extension, MATLAB displays a warning.

Additionally, if you use the `mex` function with the `-output` switch to create a MEX-file with a `.dll` extension, MATLAB displays a warning. If you use the `-output` switch to name a MEX-file, you do not need to provide a file extension. MATLAB automatically appends the appropriate extension. For example, the following command creates a MEX-file named `newtest.mexw32`:

```
mex mytest.c -output newtest
```

Compatibility Considerations

You must recompile MEX-files with a `.mexw32` file extension. This is the default of the `mex` command.

MEX-Files Must Be Recompiled When -largeArrayDims Becomes Default MEX Option

In a future version of MATLAB, the default `mex` command will change to use the large-array-handling API. This means the `-largeArrayDims` option will be the default. For information about migrating your MEX-files to use the large-array-handling API, see the Technical Support solution `1-5C27B9`.

Compatibility Considerations

In the near future you will be required to update your code to use the new API and to recompile your MEX-files. You should review your source MEX-files and `mex` build scripts.

New Compiler Support

MATLAB Version 7.7 (R2008b) supports these new compilers for building MEX-files:

Microsoft Windows 64-bit and 32-bit Platforms

- Microsoft Visual Studio 2008 Express Edition

Compiler Support to Be Phased Out

The following compilers are supported in Version 7.7 (R2008b), but will not be supported in a future version of MATLAB.

Windows (32-bit) platform

- Intel Visual Fortran Version 9.1
- Microsoft Visual Studio .NET Version 7.1
- Open Watcom Version 1.3

Windows (64-bit) platforms

- Intel Visual Fortran Version 9.1

Solaris SPARC (64-bit) platform

- Sun Studio 11 cc / CC Version 5.8
- Sun Studio 11 f90 Version 8.2

Use mxDestroyArray to Release Memory for mxArray

The documentation for the `mxSetCell`, `mxSetField`, and `mxSetFieldByNumber` functions in the MATLAB C and Fortran API incorrectly instructs customers to use `mxFree` to release memory for any `mxArray` returned by `mxGetCell`, `mxGetField`, or `mxGetFieldByNumber`.

Compatibility Considerations

The correct function to use to release memory for an `mxArray` is `mxDestroyArray`. Calling `mxFree` on an `mxArray` only frees the array header, but does not actually free the data itself and can result in a memory leak.

To help diagnose this problem, MATLAB issues a warning if calling `mxFree` on an `mxArray` could cause memory corruption. In future versions of MATLAB, this condition might result in a segmentation violation.

New Function Displays Information about MEX Compiler Configurations

The `mex.getCompilerConfigurations` function displays information about the selected compiler and associated switches and options that MATLAB uses to build a MEX-file. The selected compiler is the one you choose when you run the `mex -setup` command. For more information, see Building MEX-Files.

New Functions to Catch Errors in MEX-Files Replace mexSetTrapFlag

Two new MEX library functions have been added to the MATLAB C and Fortran API.

The `mexCallMATLABWithTrap` function, like `mexCallMATLAB`, lets you call MATLAB functions from within a MEX-file. In addition, `mexCallMATLABWithTrap` lets you catch, or trap, errors. Using this function for exception handling is more flexible that using `mexCallMATLAB` with the `mexSetTrapFlag` function.

Likewise, the `mexEvalStringWithTrap` function adds error handling to the `mexEvalString` function.

Compatibility Considerations

In the near future you will be required to update your MEX-files to remove use of the `mexSetTrapFlag` function.

“Duplicate dylib” Warning on Macintosh Systems

When compiling MEX-files on an Apple Mac OS Version 10.5 system you can ignore a warning about a duplicate library `libz.1.dylib`. The MEX-file builds properly and runs as expected. The warning message contains the following information:

```
ld: warning, duplicate dylib
```

Microsoft Visual Studio "X64 Compilers and Tools" Required for 64-bit Systems

If you use Microsoft Visual Studio with MATLAB on 64-bit systems, you must choose "X64 Compilers and Tools" when you install the following products:

- Visual Studio 2008 Express Edition
- Visual Studio 2008 Professional Edition
- Visual Studio 2005 Professional Edition

Run-Time Libraries Required for Applications Built with Microsoft Visual Studio 2008 Compiler

If you distribute a MEX-file, an engine application, or a MAT-file application built with the Visual Studio 2008 compiler, you must provide the Visual C++ run-time libraries. These files are required to run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed. For information on locating the Microsoft Visual C++ 2008 Redistributable Package (x86), containing `vcredist_x86.exe` and `vcredist_x64.exe`, consult your Microsoft documentation.

Do Not Use get or set Function to Manage Properties of Java Objects

If you want to read or update a property of a Sun Java object created in MATLAB using the Java class constructor, do *not* use the MATLAB `get` or `set` functions on the property. For example, if you create a Java object called `javaObject` that has a property called `PropertyName`, the following commands might cause memory leaks and will be deprecated in a future version of MATLAB:

```
propertyValue = get(javaObject, 'PropertyName');
set(javaObject, 'PropertyName', newValue);
```

Compatibility Considerations

In future versions of MATLAB, using `get` or `set` on Java objects to manage the properties will generate an error. The correct commands to use are:

```
propertyValue = javaObject.getPropertyName;
javaObject.setPropertyName(newValue);
```

COM Objects Might Display Different Number of Supported Events

If you use events from COM servers implementing event interface versioning, COM objects created with MATLAB Version 7.4 (R2007a), Version 7.5 (R2007b), or Version 7.6 (R2008a) might have a different number of supported events than COM objects created with MATLAB Version 7.7 (R2008b) or any version prior to 7.4.

Compatibility Considerations

Calling the `events (COM)` function on affected types of COM server components returns a list of events for the latest [default] interface version which might be different from the list of events displayed by MATLAB Version 7.4, Version 7.5, or Version 7.6.

If an event in an older COM event interface version is no longer supported or renamed in the newer interface version, the `registerevent` function generates an error when such an event is used in an `.m` file.

R2008a

Desktop Tools and Development Environment

Desktop New Features Video

For an overview of the major new features in the MATLAB Desktop Tools and Development Environment area, watch this video demo. You can also access this and other video demos by selecting the **Demos** tab in the Help browser, and then selecting **MATLAB > New Features in Version 7.6**.

Startup and Shutdown

New features and changes introduced in Version 7.6 (R2008a) are:

Windows Platforms — Startup Changes, Including Use of My Documents/MATLAB or Documents/MATLAB Directory

On Microsoft Windows platforms, when MATLAB starts, it automatically adds the `My Documents/MATLAB` directory (or `Documents/MATLAB` on Windows Vista™) to the top of the MATLAB search path. This directory is known as the `userpath`. If you remove the `My Documents/MATLAB` (or `Documents/MATLAB`) directory from the search path and save the changes, either using the Set Path dialog box or using the `rmpath` and `savepath` functions, the MATLAB search path will not include the `userpath` directory at the next startup. On Windows platforms, the `userpath` is also the default startup directory.

Use the new function, `userpath`, to view the current value, clear the value so the directory is *not* on the search path and is *not* the startup directory, specify a different value, or reset the value to the default. For more information, see Default Startup Directory (Folder) on Windows Platforms and the `userpath` reference page.

Compatibility Considerations

In previous versions, MATLAB automatically added the `My Documents/MATLAB` directory (or `Documents/MATLAB` on Windows Vista platforms) to the search path upon startup, even if you had removed it from the path and saved your changes during the previous session.

UNIX Platforms — Startup Changes

On UNIX platforms, when MATLAB starts, it automatically adds the *`userhome`*`/Documents/MATLAB` directory to the top of the MATLAB search path. This directory is known as the `userpath`. Use the new function, `userpath`, to view the current value, clear the value so the directory is *not* on the

search path, specify a different value, or reset the value to the default. You can also specify that `userpath` be the MATLAB startup directory by setting the value for the environment variable `MATLAB_USE_USERWORK` to `1` prior to startup. For more information, see Startup Directory on UNIX Platforms and the `userpath` reference page.

Compatibility Considerations

In previous versions, no directories were added to the search path upon startup.

Macintosh Platforms — Startup Changes

On Apple Macintosh platforms, when MATLAB starts, it automatically adds the *`userhome`*`/Documents/MATLAB` directory to the top of the MATLAB search path. This directory is known as the `userpath`. Use the new function, `userpath`, to view the current value, clear the value so the directory is *not* on the search path, specify a different value, or reset the value to the default. If you start MATLAB from a shell, you can also specify that `userpath` be the MATLAB startup directory by setting the value for the environment variable `MATLAB_USE_USERWORK` to `1`.

For more information, see Startup Directory on <trademark Macintosh Platforms and the `userpath` reference page.

Compatibility Considerations

In previous versions, no directories were added to the search path upon startup.

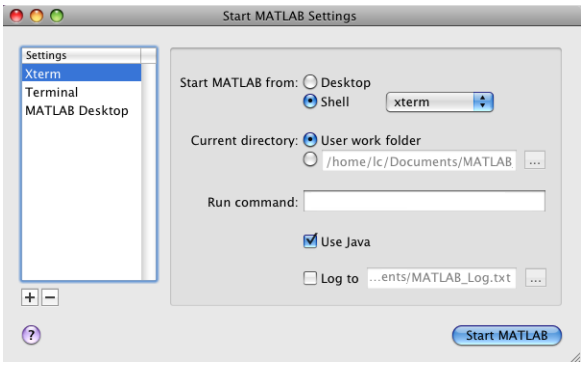
Macintosh Platforms — Define Startup Options Using New Dialog Box

On Apple Macintosh platforms, you can specify startup options using the new Start MATLAB Settings dialog box. The first time you start MATLAB Version 7.6 (R2008a), the Start MATLAB Settings dialog box opens automatically. It does not open automatically on subsequent startups; to open the dialog box, double-click `Start MATLAB Settings`, located in the same directory as the MATLAB application.

You can set these options and others:

- How you will interact with MATLAB (desktop or specified shell)
- The startup directory; the default is *`userhome`*`/Documents/MATLAB`
- Statement MATLAB runs upon startup

You can create and save multiple startup options files, each with different settings. For more information, see Startup Options.



Macintosh Platforms — Run Startup Diagnostics

Upon startup on Macintosh platforms, MATLAB automatically runs diagnostics to ensure your system has the required X11 and Sun Microsystems Java applications. If there are no problems, MATLAB starts as expected. If there is a problem, MATLAB displays a dialog box informing you of the problem and how to address it. You can run these diagnostics manually from the Start MATLAB Settings dialog box, using items in the **Diagnostics** menu.

Updated Version of JVM Software on Solaris Platform

MATLAB is now using Sun Microsystems JVM Version 6 on the Sun Microsystems Solaris platform.

Compatibility Considerations

If you use a specific version of Sun Microsystems Java with MATLAB on the Solaris platform, this change might impact your work.

Changes to Abnormal Termination Process

When MATLAB encounters a serious problem, such as a segmentation violation, a dialog box opens to notify you about the problem. From the dialog box, you can close MATLAB, or try to save your work in progress before closing.

If you try to save your work in progress, be aware that MATLAB is in an unreliable state and you should exit MATLAB as soon as you finish saving your work. The Command Window displays the message `Please exit and restart MATLAB` to the left of the prompt, which reminds you to discontinue use. For more information, see Abnormal Termination.

Compatibility Considerations

In previous versions, when MATLAB encountered a serious problem, an error message appeared in the Command Window that instructed you to close MATLAB.

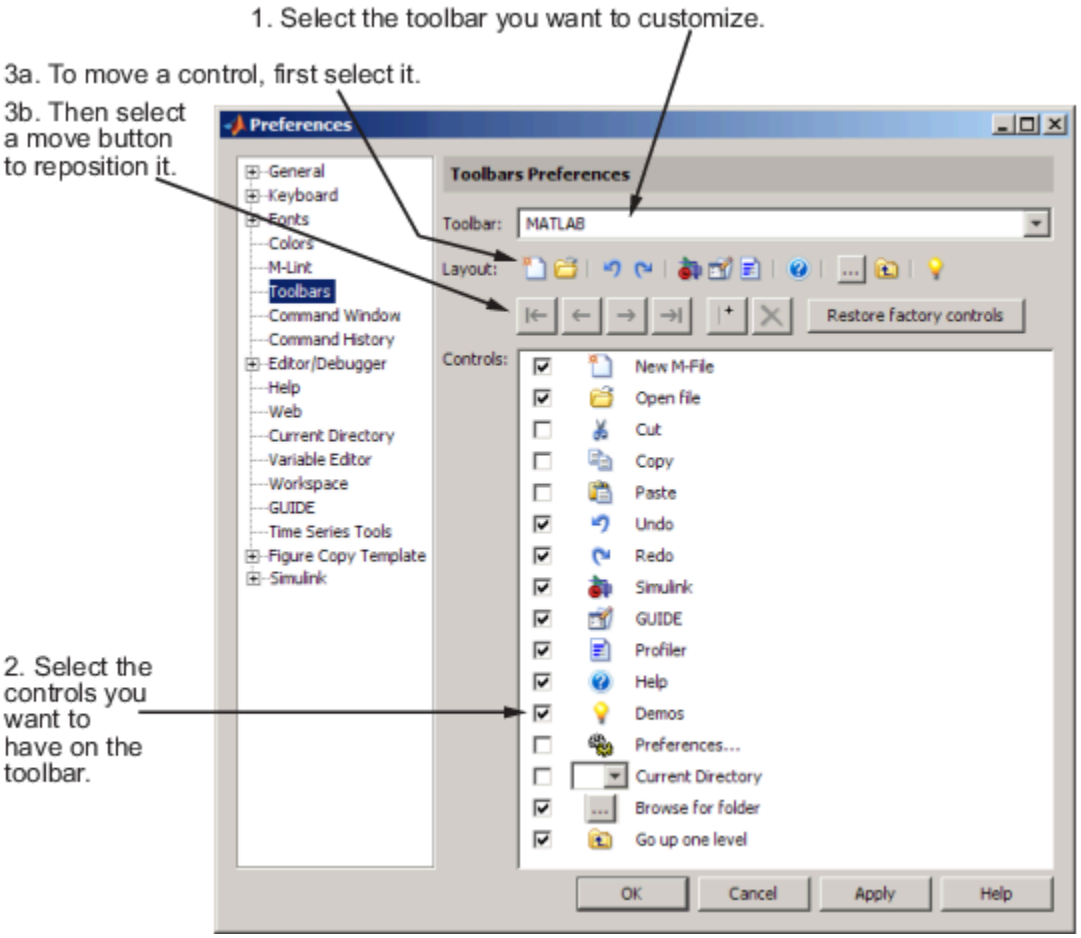
With multithreaded computation enabled, a platform-specific dialog box appeared, from which you immediately closed MATLAB.

Desktop

New features and changes introduced in Version 7.6 (R2008a) are:

Customize the MATLAB Desktop and Editor Toolbars

Rearrange, add, or remove buttons and other controls from the MATLAB desktop or Editor toolbars using **File > Preferences > Toolbars**. Alternatively, right-click a toolbar and select **Customize** from the context menu.



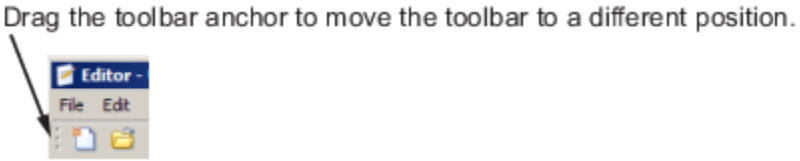
For more information, see Toolbar Customization.

There are new buttons you can add to the MATLAB desktop toolbar:

- Preferences, , which displays the Preferences dialog box, open to the pane last used
- Demos, , which displays the listing of Demos in the Help browser

You can also add Save All and Save As buttons to the Editor toolbar.

You can change the position of the toolbars within a tool, for example, putting both the Editor and Editor cell mode toolbars next to each other instead of stacked. To move a toolbar, grab the anchor for a toolbar, then drag the toolbar to the new location.



Clear a Browser with New Method

Clear an open browser in MATLAB with a new `close` method. For example, open a browser to display the MathWorks Web site by running `[stat,h1]=web('http://www.mathworks.com')`. Then `close(h1)` clears `mathworks.com` from that browser window. For more information, see the reference page for the `web` function.

Manage Your License

You can use new licensing features to perform license management activities, such as activating your license, deactivating your license, or updating your license. You can also visit the License Center at the MathWorks Web site to perform other license-related activities. Access the features by selecting **Help > Licensing**.

Check for Updates Feature Enhanced

When you select **Help > Check for Updates**, the dialog box now allows you to see the latest versions for all MathWorks products, or just those you install. You can also access release notes for each product from the dialog box. For more information, see Check for Software Updates.

Running Functions — Command Window and History

Command History Preference — Default Value Changed

The default value for the Command History preference, **Save after *n* commands**, is now 1. This allows you to more easily rebuild your state in MATLAB if MATLAB terminates abnormally, such as after a power failure. For more information about this preference, see Command History Preferences.

Compatibility Considerations

Previously, the default value for the **Save after *n* commands** preference was 5.

Help

Preferences for Help on Selection

When you click a function name in the Command Window or Editor, and then press **F1** or select **Help on Selection** from the context menu, the help appears in a pop-up window by default. Now, you can specify that the help appear in the Help browser rather than in a pop-up window via the new **Help on Selection** preference. For more information, see Help Preferences.

Slight Reordering of Products in Help Browser

In the Help browser, the order of some documentation names in the **Contents** pane and the Product Filter preference dialog box has changed slightly. Documentation now appears in this order:

- Release Notes (general)
- Installation
- MATLAB
- Toolboxes and other products based on MATLAB, arranged alphabetically
- Simulink

- Blocksets and other products based on Simulink, arranged alphabetically
- Link and target products, arranged alphabetically

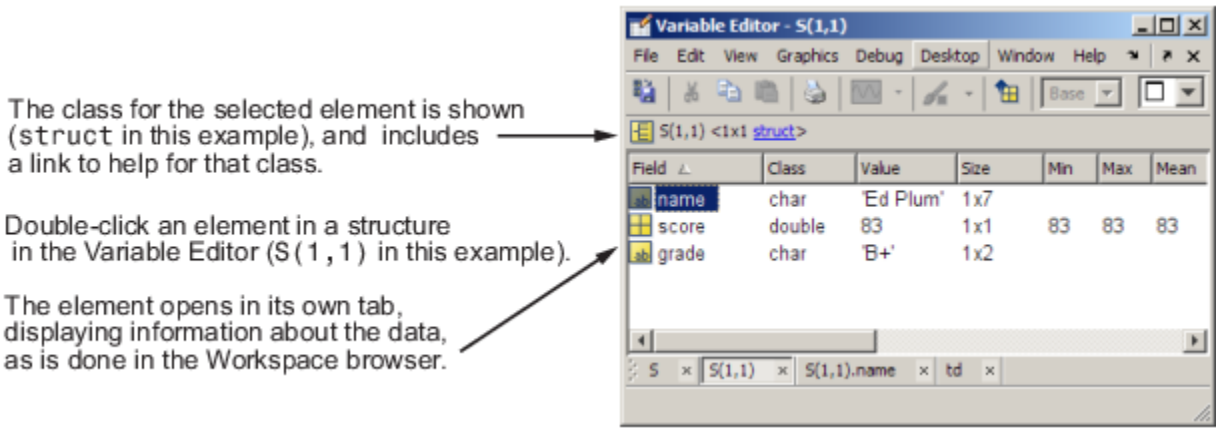
Workspace, Search Path, and File Operations

New features and changes introduced in Version 7.6 (R2008a) are:

Array Editor Renamed to Variable Editor; Offers Enhanced Support for Structures and Classes

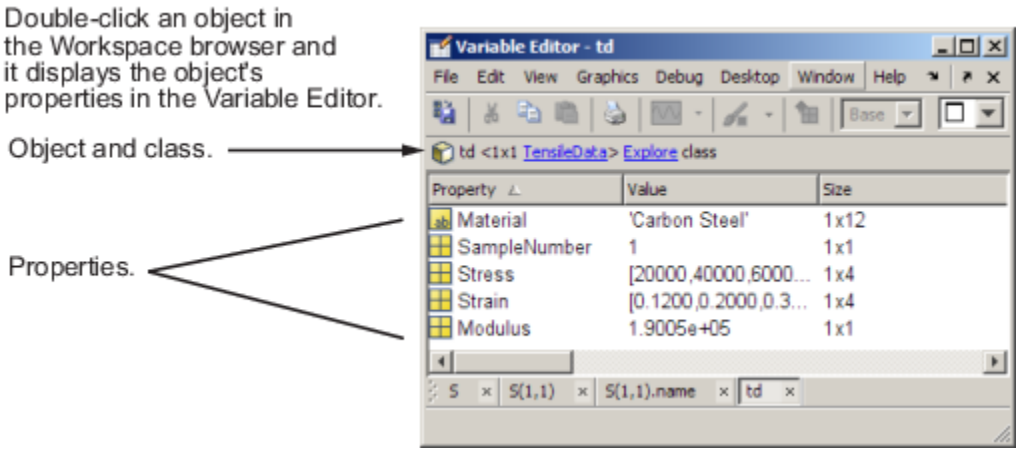
The Array Editor has been renamed to the Variable Editor, which better reflects its support for non-array data such as structures and properties.

The Variable Editor now reports, just below the toolbar, the class and size of the selected variable. For many classes, there is also a link to help for the class. To view a structure in the Variable Editor, double-click one of its elements. The resulting display is much like the display of that element in the Workspace browser, showing the Class, Value, Size, Min, Max, and other information. For more information about the Variable Editor, see Viewing and Editing Workspace Variables with the Variable Editor.



Use the data brushing button on the Variable Editor toolbar  to mark observations on graphs and then remove or save them to new variables. For more information, see "Data Brushing for Graphs and Linked Variables" on page 29-24.

The Variable Editor supports the new MATLAB class system. Double-click an object in the Workspace browser, and it opens in the Variable Editor. The Variable Editor displays the object, the class, and the properties of the object.



Compatibility Considerations

In previous versions, the Variable Editor was called the Array Editor. Starting in R2008a, MATLAB Version 7.6, the tool will be referred to as the Variable Editor.

Search Path — Changes to User Portion

By default, MATLAB adds a directory to the top of the search path upon startup, known as the `userpath` directory. By default, its value is `Documents/MATLAB` on Windows platforms, or `My Documents/MATLAB` on Windows Vista platforms. On UNIX and Macintosh platforms, the default directory is *`userhome`*`/Documents/MATLAB`. Use the new function, `userpath`, to specify a different directory, clear the `userpath` value, or reset it to the default value. On UNIX and Macintosh platforms, you can specify additional directories for MATLAB to add to the top of the search path upon startup using the `MATLABPATH` environment variable. When you remove the `userpath` portion of the search path, it clears the value for `userpath` and might impact the startup directory. For related information, see “Startup and Shutdown” on page 29-2. For details see Locations for Storing Your Files and the `userpath` reference page.

Compatibility Considerations

In previous versions, on Windows platforms, MATLAB added the default directory to the search path upon startup, even if you removed it and saved the changes to the path. On UNIX and Macintosh platforms, MATLAB did not add a directory to the search path on startup.

New Context Menu Options in Current Directory Browser

The context menu, which you access by right-clicking anywhere within the Current Directory browser, provides these three new options for creating M-files in the current directory:

- **New > Blank M-File**

  Creates an empty M-file
- **New > Function M-File**

  Creates an M-file with a template for writing an M-file function
- **New > Class M-File**

Creates an M-file with a template for writing an M-file class definition

Compatibility Considerations

The **New > M-File** option is replaced by the **New > Function M-File** option, which has the same effect.

File and Directory Comparisons Tool

The File Comparisons tool is now called the File and Directory Comparisons Tool. In addition to enabling you to compare lines in two text files, it now enables you to:

- Compare variables in two MAT files
- Determine whether the contents of two binary files are the same
- Compare two directories to determine which file names are unique to each directory
- Compare two directories to determine if files with the same name in each directory have the same content

See Comparing Files and Folders for details.

Tuning and Managing M-Files

Profiling — Setting Intel Multi-Core Processors

If your system uses Intel multi-core chips, and you plan to profile using CPU time, set the number of active CPUs to 1 before you start profiling. See Intel Multi-Core Processors — Setting for Most Accurate Profiling on Windows Systems for details.

Editing and Debugging M-Files

New features and changes introduced in Version 7.6 (R2008a) are:

- “Stand-Alone Editor No Longer Provided” on page 29-9
- “Run/Continue Button Now Two Separate Buttons” on page 29-10
- “Evaluate Entire File Button Off Toolbar by Default” on page 29-10
- “TLC and XML Syntax Highlighting Supported” on page 29-10
- “Code Folding Enhanced to Support More Language Constructs” on page 29-10
- “mlint Function Uses Preference Settings when Java Software is Available” on page 29-11
- “New M-Lint Warning Related to the MException Class” on page 29-11
- “dbstop and dbclear Functions — Option to Specify File Not on Path” on page 29-12
- “edit Function Can Create New File in Existing Subdirectory” on page 29-12
- “Nest Cells for Rapid Code Iteration; Includes Changes to Cell Highlighting” on page 29-12

Stand-Alone Editor No Longer Provided

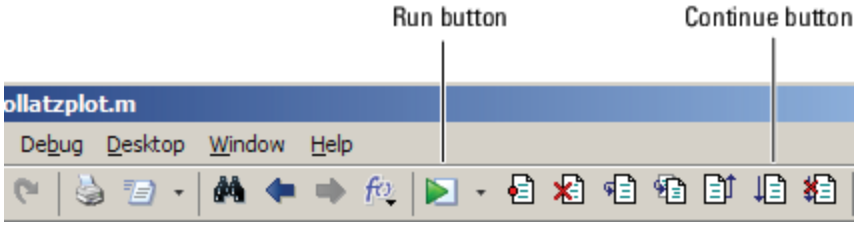
The MATLAB stand-alone Editor (`meditor.exe`) is no longer provided. Instead of the stand-alone Editor, you can use the MATLAB Editor.

Compatibility Considerations

Some users have preferred the stand-alone Editor to the MATLAB Editor because of slightly better startup performance and because it does not require a MATLAB software license. For those situations, you can use any text editor you have, such as UltraEdit or Emacs.

Run/Continue Button Now Two Separate Buttons

The Continue button is now separate from the Run button. Previously, the Run button served as both the Run and Continue button.



Compatibility Considerations

You now use the Run button to execute a run configuration and the Continue button to continue execution of an M-file after a breakpoint during debugging. See Run MATLAB Files in the Editor and Step Through a File for details.

Evaluate Entire File Button Off Toolbar by Default

The Evaluate Entire File button, , is no longer on the Editor Cell Mode toolbar by default.

Compatibility Considerations

Previously, the Evaluate Entire File button was on the Editor Cell Mode toolbar by default. You can put the Evaluate Entire File button back on the toolbar by customizing it. See “Customize the MATLAB Desktop and Editor Toolbars” on page 29-4 for more information.

TLC and XML Syntax Highlighting Supported

You can specify preferences for TLC and XML syntax highlighting in the Editor/Debugger Language Preferences panel by selecting **File > Preferences > Editor > Language** and then, in the **Language** drop–down menu choosing **TLC** or **XML/HTML**.

Click **Help** in the Preferences dialog box for more information.

Code Folding Enhanced to Support More Language Constructs

You can enable code folding for all these programming constructs:

- Blocks of comments
- Cells used for rapid code iteration and publishing
- Class code
- Class enumeration blocks

- Class event blocks
- Class method blocks
- Class properties blocks
- For and parfor blocks
- Function and class help
- Function code
- If/else blocks
- Switch/case blocks
- Try/catch blocks
- While blocks

Prior to MATLAB Version 7.6 (R2008a), code folding was supported for function code and function help only.

See Code Folding — Expand and Collapse Code Constructs for details.

mlint Function Uses Preference Settings when Java Software is Available

When Sun Microsystems Java software is available, the `mlint` function honors the M-lint preferences that you specify using the M-Lint Preferences dialog box (**File > Preferences > M-Lint**).

In addition, the `'−config=settings.txt'` option to the `mlint` function enables you to override the current default M-Lint preferences settings with a settings file that you previously created and saved using the M-Lint Preferences dialog box. If you use the `'−config=settings.txt'` option, you must specify the full path to the file. If you prefer that `m-lint` ignore all M-Lint preferences and use the factory default settings instead, specify the `'−config=factory'` flag. See `m-lint` for details.

Compatibility Considerations

Previously, the `mlint` function always used the factory default settings, regardless of the M-Lint preferences that you set. To restore that behavior, use the `'−config=factory'` flag on the `mlint` function.

New M-Lint Warning Related to the MException Class

MATLAB Version 7.6 (R2008a) adds a new M-Lint warning related to the `MException` class. This warning, along with two M-Lint warnings added in MATLAB Version 7.5 (R2007b), intentionally make it difficult for you to completely ignore an error without receiving an M-Lint warning. It is a best practice to check error information even when an error is expected or frequent, so that you can rule out unexpected situations.

The three messages are as follows—the first is the one added in MATLAB Version 7.6 (R2008a):

- `LASTERR and LASTERROR are better replaced by an identifier on the CATCH block. See doc CATCH.`
- `The value assigned here to variable 'x' might never be used.`

  This message appears when the code contains a `catch` statement that is never used.
- `TRY statement without a CATCH.`

For example, suppose you want to read options from a file, `options.txt`, but it is acceptable if that file is not present. You might write the following code, expecting the `read_options` program to throw an error if the file is not present:

```
options = {};
try
    options = read_options( 'options.txt' );
end
```

The problem with the preceding code is that the file might be present, but its permissions may prevent the program from reading it. The program ignores the file, and potentially confuses the user, who knows the file is there. Better code for accomplishing the task is as follows, which assigns a structure value to the variable `err` if an error is thrown in the `try` block. The structure value contains information about the error that was thrown.

```
try
    options = read_options( 'options.txt' );
catch err
    if strcmp( err.identifier, 'Program:ReadOptions:NoOptionsFile' )
        options = {};
    else
        rethrow( err );
    end
end
```

Using this code, if a problem other than the “file is missing” error occurs, MATLAB reports the error to the user. For instance, MATLAB reports an error if the file format is incorrect, or if the file has the wrong permissions.

If you feel comfortable ignoring the errors completely, it is probably best to use the `try` statement with no `catch` statement, and suppress the M-Lint warnings that result. You can suppress the warnings through the M-Lint preferences or by placing the `%#ok` pragma at the end of the line that triggers the message. However, The MathWorks suggests that if you suppress an M-Lint message, you include a comment in your code indicating why you think it is appropriate to ignore the message.

For more information about the `MException` class, see the Error Handling section in the MATLAB Programming Fundamentals documentation.

dbstop and dbclear Functions — Option to Specify File Not on Path

The `-completenames` option to the `dbstop` and `dbclear` functions enables you to set and clear breakpoints, respectively, for M-files that are not on the search path in MATLAB. See `dbstop` and `dbclear` for details.

edit Function Can Create New File in Existing Subdirectory

The edit function now allows you to specify a file that is not in the current directory. If the file does not exist, the edit function creates it in the directory you specify. However, the directory, must exist; the edit function will not create a directory for you. See `edit` for details.

Nest Cells for Rapid Code Iteration; Includes Changes to Cell Highlighting

You can nest cells in an M-file, including within functions and control statements, such as `for` loops and `if-then` blocks. This gives you greater control over how a published document appears. This nesting ability also enables you to evaluate subsections of code on a finer grain. See “Nest Cells for Finer Control” on page 29-14 for details.

Compatibility Considerations

With the introduction of nested cells, cells definitions result in cell highlighting that looks different from previous releases. See Define Code Cells for details.

Publishing M-Files

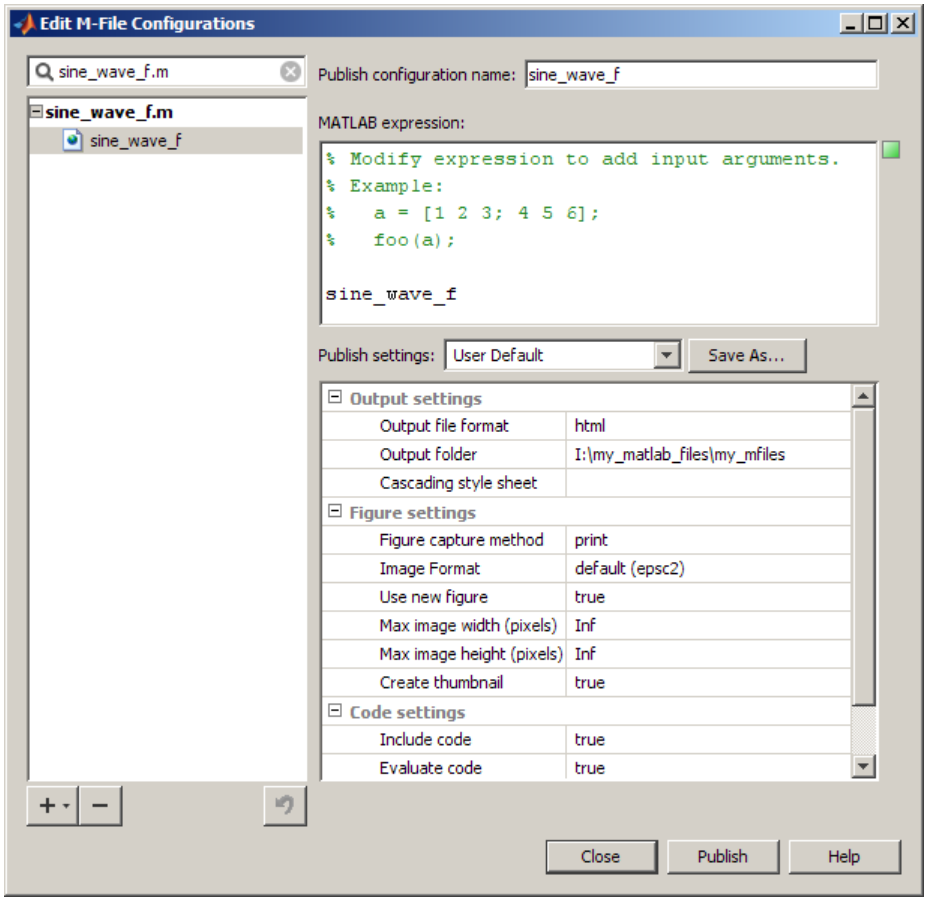
New features and changes introduced in Version 7.6 (R2008a) are:

Publish Functions and Scripts Using Publish Configurations; Includes Replacement of Publishing Preferences

In the Editor, you can now do the following when publishing M-file code:

- Specify code that you want MATLAB to evaluate before publishing the code, including input arguments for functions
- Specify publishing settings, such as an output directory and file format, that you can save and reuse as a group

To create a publish configuration, first open an M-file in the Editor. Then, select **File > Publish Configurations for *`filename`* > Edit Publish Configurations for *`filename`***. In the resulting Edit M-File Configurations dialog box, modify the **MATLAB expression**, specify **Publish settings**, and name the publish configuration. For more information, see Creating a Publish Configuration for a MATLAB File.



Compatibility Considerations

Previously, preferences for publishing and publishing images were available on the Preferences dialog box, which you accessed by selecting **File > Preferences > Editor/Debugger** and then choosing the **Publishing** or **Publishing Images** node. Now you set these preferences when you create or update a publish configuration by using the **Publish settings** options on the Edit M-file Configurations dialog box.

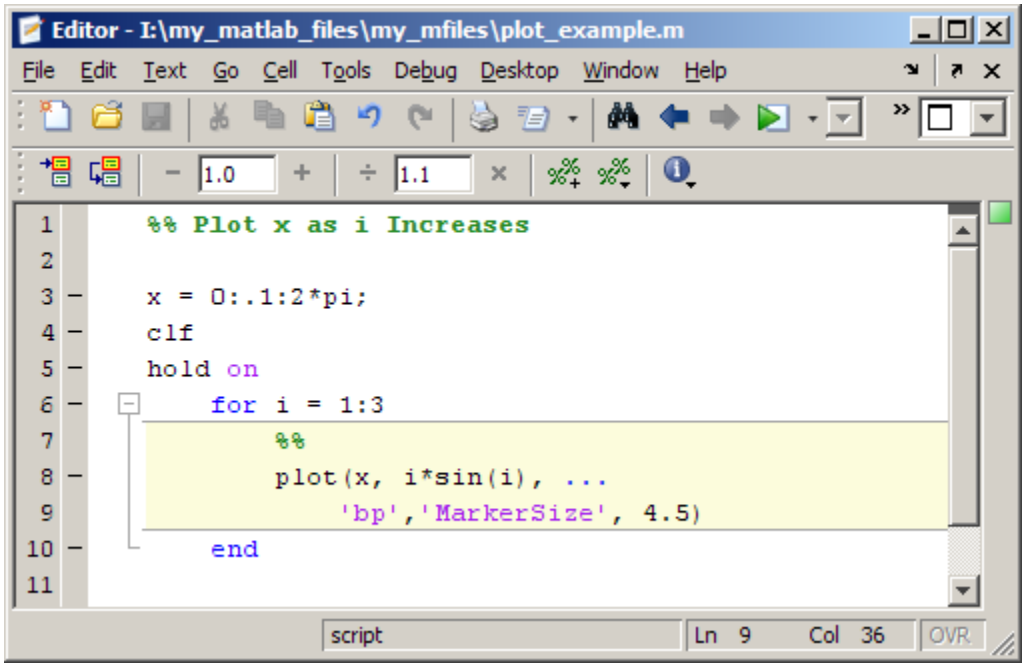
The Edit M-file Configurations dialog box continues to provide support for creating configurations that enable you to run an M-file. These are now called run configurations to differentiate them from publish configurations.

Nest Cells for Finer Control

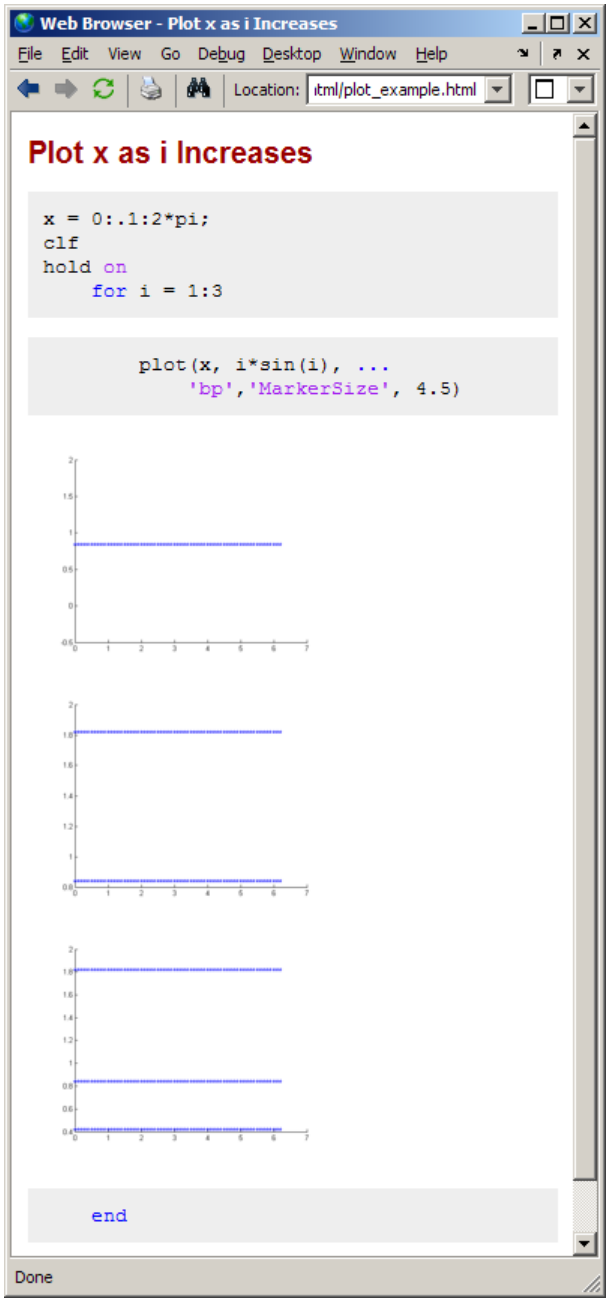
You can nest cells in an M-file, including within functions and control statements, such as `for` loops and `if-then` blocks. This gives you greater control over how a published document appears. This nesting ability also enables you to evaluate subsections of code on a finer grain when using rapid code iteration.

You can insert white space before the double percent (%%) characters that specify a cell break (which is also referred to as a cell divider). This helps to improve readability of the M-file when the cell break is within indented code. In prior releases, the %% characters had to be in the first column of the code.

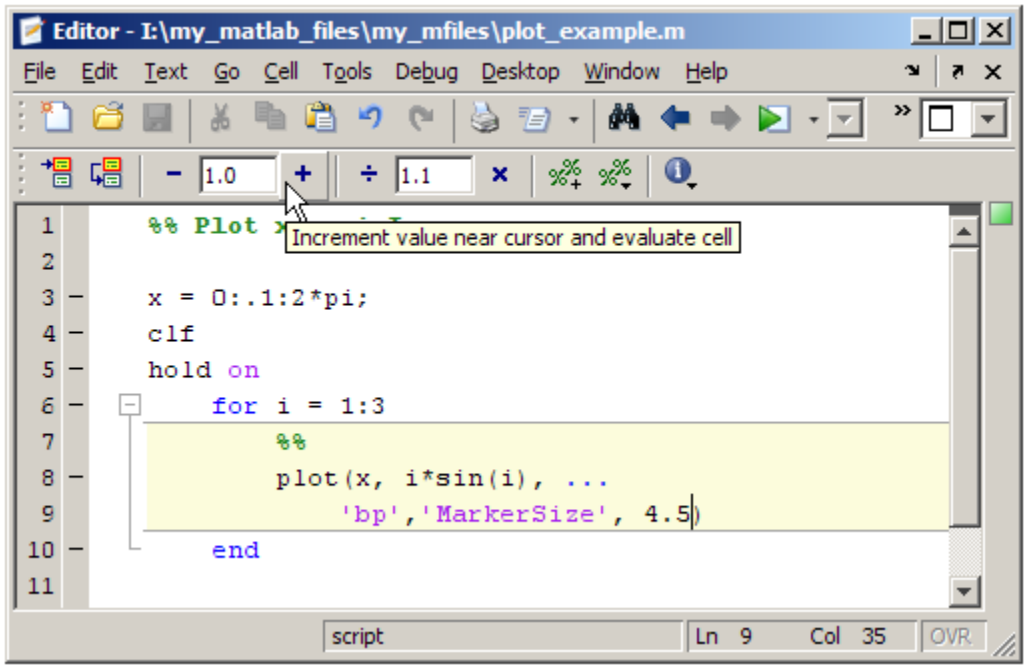
The following image shows a simple example of an M-file with nested cells.



If you publish the file to HTML (reducing the size of the images), the cell break nested within the `for` loop causes MATLAB to publish each iteration of the `for` loop as it evaluates the code in the loop:



Similarly, the cell break within the `for` loop enables you to run the M-file and experiment with the marker size value without the need to save the file between adjustments:



For more information see Mark Up MATLAB Code for Publishing and Evaluate Subsections of Files Using Code Cells.

Compatibility Considerations

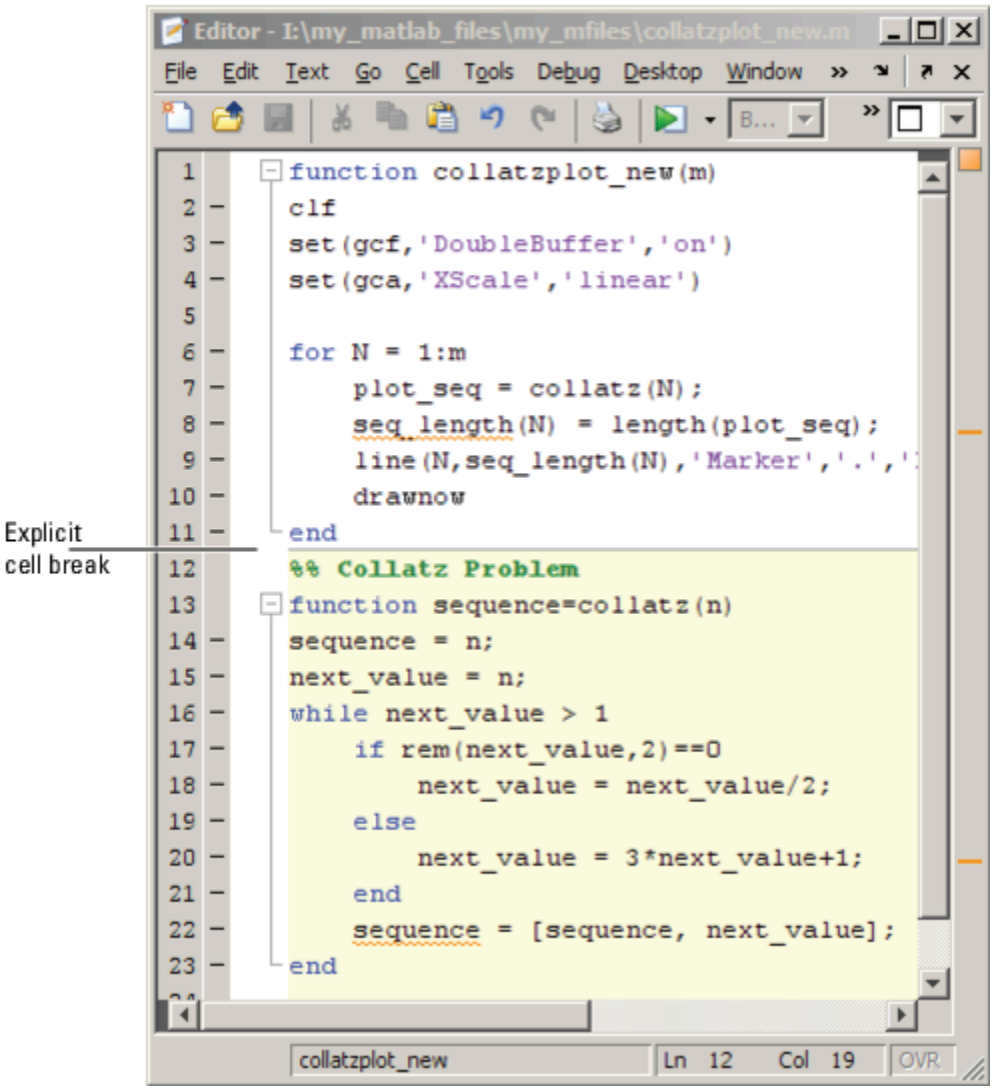
In prior releases, the cell break characters (%%) had to be in the first column of the code for the Editor to recognize the characters as a cell break. This is no longer true, the Editor now recognizes these characters as a cell break regardless of the amount of white space that precedes them.

Furthermore, with the introduction of nested cells, inserting cell breaks in this release has different effects than in the previous release. In the previous release, if you inserted a cell break before a subfunction declaration, MATLAB created two cells; one above the cell break and one below it. Now, if you insert a cell break, MATLAB also inserts implicit cell breaks.

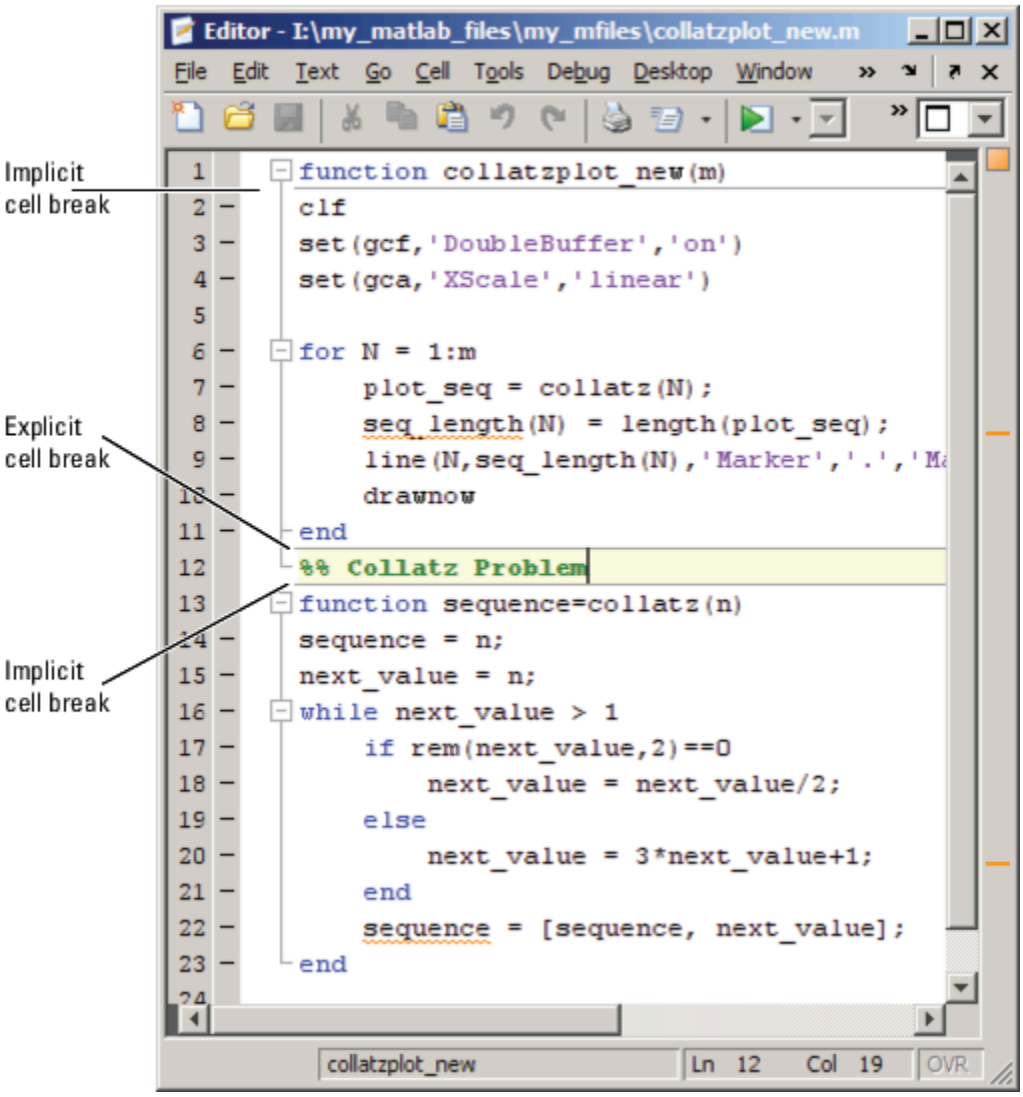
In the example shown, it inserts two implicit cell breaks in the subfunction where you inserted the explicit cell break; one on the first line of the subfunction and one on the last line of the subfunction. This results in three cells: one containing the entire file, one containing the `collatzplot_new` function, and one containing just the Collatz Problem cell title.

Nested cells are introduced to support function publishing.

Version 7.5 (R2007b)



Version 7.6 (R2008a)



See Nested Code Cells for details.

Publish Button Moved

The Publish button,  , is now located on the Editor toolbar.

Compatibility Considerations

Previously, the Publish button was located on the Editor Cell Mode toolbar. Now you find it on the Editor toolbar.

Publish Trademark Symbols

If the comments in your M-file include trademarked terms, you can format the comment to produce a trademark symbol (™) or registered trademark symbol (®) in the published output. See Trademark Symbols for details.

Specifying Code for MATLAB Software to Evaluate with the publish Function

Use the `codetoEvaluate` option to the `publish` function to specify code that you want MATLAB software to evaluate when it publishes an M-file. By default, this is the code in the M-file. However, if you want, you can use this option to specify additional code or alternative code for MATLAB to evaluate. For example, you might want MATLAB to evaluate code that calls the M-file that you are publishing.

stopOnError Option No Longer Available with publish Function

The `stopOnError` option is no longer available for the `publish` function. MATLAB software will always stop when an error occurs, unless you add code to handle the error.

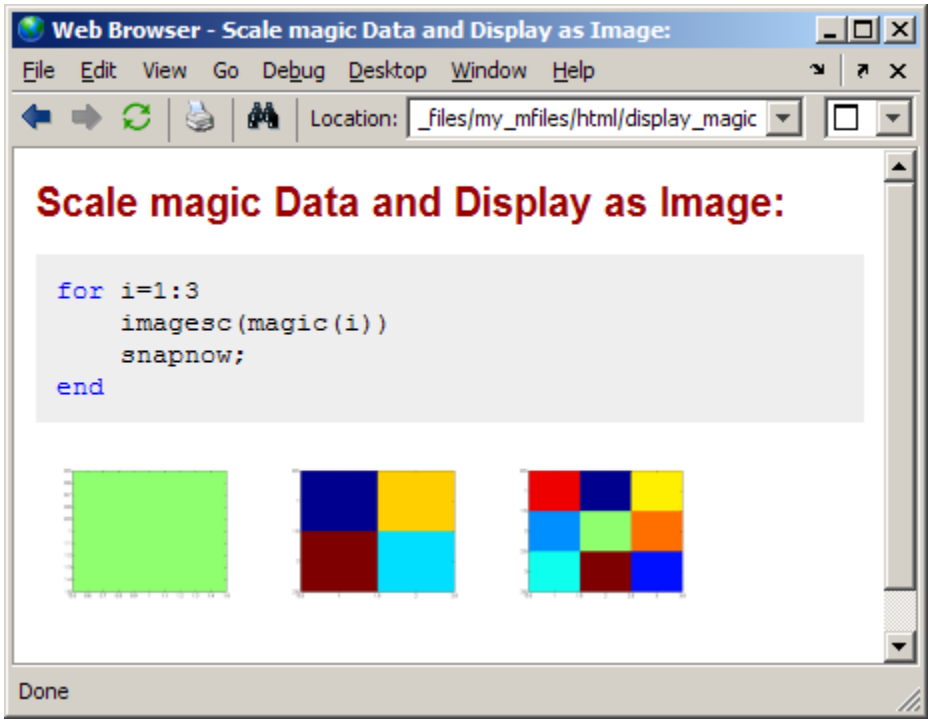
Compatibility Considerations

To have MATLAB continue processing code when an error occurs, handle the error using a `try-catch` statement. For more information, see The try-catch Statement.

Include Snapshot of M-file Output in Published Document

You can include snapshots of output that an M-file generates within a published document. Select **Cell > Insert Text Markup > Force Snapshot**. This menu option inserts the `snapnow` function into your M-file code. This is particularly useful when you have code that generates numerous images that you want to include in the published document. See Force a Snapshot of Output for details.

The following image, for example, shows a published document that uses this feature (with the size of the images reduced). Notice that the published images appear after the `for` loop that generates them.



Internationalization

Locale Information Added to MATLAB Documentation

Information about using locale in MATLAB can be found in Internationalization in the Desktop Tools and Development Environment documentation.

Changes to Locale Database

Windows Platform Changes

On Microsoft Windows systems, users can select the Pashto language with the Afghanistan country code. This locale setting is `ps_AF.1256`.

Macintosh OS X Platform Changes

On Apple Macintosh OS X systems, for users selecting the Chinese language and the China country code, the locale setting is `zh_CN.gb2312`. The previous setting was `zh_CN.GBK`.

Mathematics

Upgrade to BLAS Libraries

MATLAB software now uses new versions of the Basic Linear Algebra Subroutine (BLAS) libraries. For Windows, Intel Mac, and Linux platforms, MATLAB software supports the Intel Math Kernel Library (MKL) version 9.1. For the Solaris platform, MATLAB software uses the Sun Performance Library from Sun Studio 12.

Upgrade to LAPACK Library

MATLAB software now uses Version 3.1.1 of the Linear Algebra Package (LAPACK) library.

More Multithreaded Support For Elementwise Math Functions With Warnings

Multithreaded support has been added to elementwise math functions that may generate warnings: `rdivide`, `ldivide`, `log`, `log2`, and `rem`.

New Algorithms for ldl, logm, and funm Functions

The `ldl`, `logm`, and `funm` functions include new algorithms based on recent numerical methods research.

Functions and Properties Being Removed

Function or Property Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`betacore`	Errors	`betainc`	Replace all existing instances of `betacore` with `betainc`.
`colmmd`	Errors	`colamd`	Replace all existing instances of `colmmd` with `colamd`.
`flops`	Errors	None	Remove all existing instances of `flops`. With the incorporation of LAPACK in MATLAB version 6, counting floating-point operations is no longer practical.
`symmmd`	Errors	`symamd`	Replace all existing instances of `symmd` with `symamd`.
`quad8`	Errors	`quadl`	Replace all existing instances of `quad8` with `quadl`.
`table1`	Errors	`interp1` or `interp1q`	Replace all existing instances of `table1` with `interp1` or `interp1q`.

Function or Property Name	What Happens When You Use Function or Property?	Use This Instead	Compatibility Considerations
`table2`	Errors	`interp2`	Replace all existing instances of `table2` with `interp2`.
`bessela`	Errors	`besselj`	Replace all existing instances of `bessela` with `besselj`.
`beta` using three input arguments	Errors	`betainc` using three input arguments	Replace all existing instances of `beta` using three input arguments with `betainc` using three input arguments.

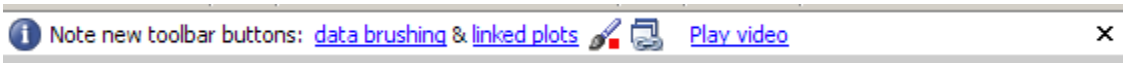
Data Analysis

Data Brushing for Graphs and Linked Variables

This release introduces two new interactive tools for data exploration:

- **Data brushing** — For marking observations on graphs, allowing you to remove or save them to new variables
- **Data linking** — For connecting graphs with data sources (workspace variables) to automatically and interactively update them

In addition, figure windows have a new banner, called the linking and brushing message bar. By default, when you plot data into a figure (i.e., add axes), an informational banner appears across top of the figure that looks like this.



In the figure's message bar, click the first two links to read about these new tools. Click the **Play video** link to open a nine-minute video tutorial about the tools in a browser window (the video also describes new GUI-building features.) To dismiss the banner, click the **X**. Once you do, the banner only reappears on subsequent plots if you select **Show linking and brushing message bar** in the MATLAB Preferences Confirmation Dialogs panel.

Note The linking and brushing message bar can obscure a plot's title. Also, if you do not dismiss the message bar, it is visible in images of figures captured with `getframe`, but it does not print. See “Compatibility Considerations” on page 29-27 for details.

Use data brushing when you want to isolate observations in a 2-D or 3-D graph for separate analysis, or to remove outliers or noisy data points. Data brushing can be applied to most graphs (some plot types do not support brushing). Data brushing is an exclusive, persistent mode. That is, when using it, you cannot use other figure tools, but the results of brushing data persist when you select a different tool or no tool.

Use data linking to make plots dynamically respond to changes in the variables they plot. Data linking applies to most graphs with identifiable data sources and operates at the figure level. Data linking is not modal and persists until you toggle it off or the connection between a plot and its data sources is broken.

The two tools work smoothly together and with the Variable Editor to visually highlight brushed observations and the data values they represent:

- Brushing data-linked observations on a graph highlights them on other graphs that display them.
- Brushing highlights values in the Variable Editor when a brushed variable is displayed there.

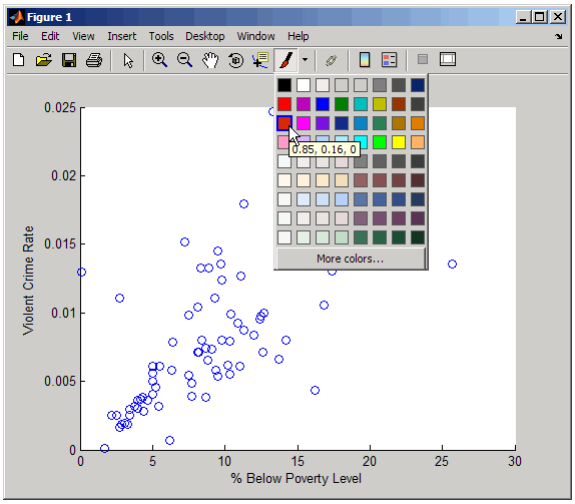
- Using the Brush tool in the Variable Editor highlights values you brush that appear in linked plots.
- Changing values of variables causes linked graphs displaying them to update with the changes.
- Clearing variables disconnects them from all linked figures displaying graphs of them

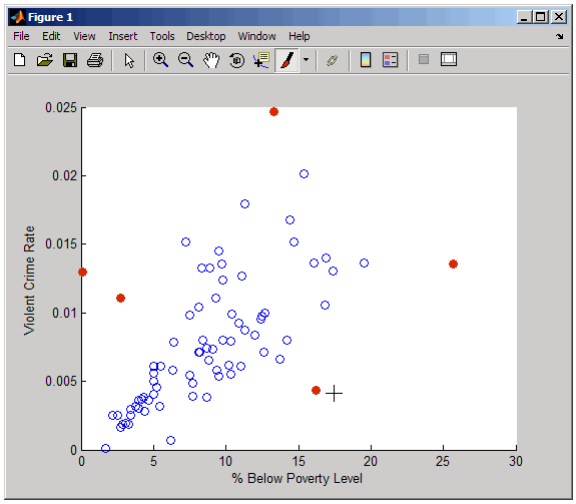
You can modify variables from the command line, the Variable Editor, or with M-files. When used within functions, data linking operates in the function's workspace, not the base workspace. This is also the case when debugging.

Data Brushing Tool

All figure windows that contain axes now include a data brushing tool (the Brush/Select data icon ) that lets you enter and exit brushing mode and select a color with which to brush observations. The tool draws selection rectangles (in 2-D plots) or prisms (in 3-D plots) and permits you to select discontiguous regions and negate previously brushed observations. Undo is also supported.

The Brush/Select data tool is a “split button” control with a brush icon on the left and a drop-down color palette on the right. When you depress the brush icon, you are in brushing mode; all data observations you select are highlighted with the current brush color. The figures below illustrate these operations.





If you leave data brushing mode to zoom, pan, or edit the plot, all brushed observations remain highlighted. You can then reenter brushing mode and pick up where you left off. Brush marks are not preserved when you save a figure and reopen it from the FIG-file, however.

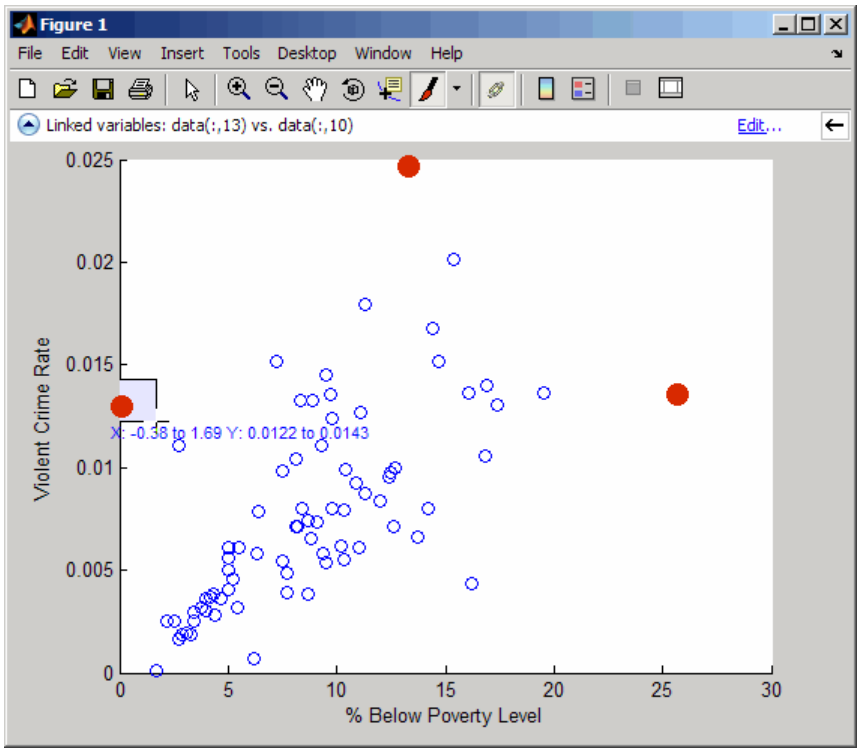
Data Brushing API

Use the `brush` function to turn brushing on and off, and to select a color for brushing graphs. You can change brush colors on the fly with either the API or with the Brush tool.

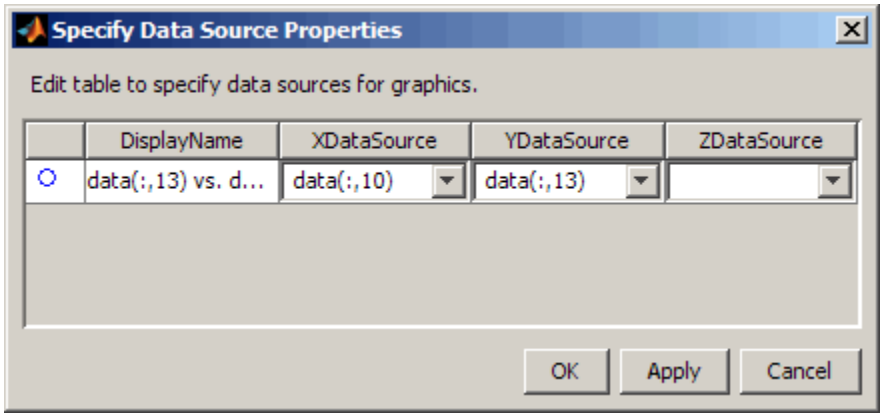
Data Linking Tool

All figure windows that contain axes now include a data linking tool (the Linked Plots button ) to toggle linked mode on and off (the default). When you toggle it on, an information bar appears

underneath the lowest toolbar on the figure, as shown below. It displays what variables are linked to each series (data sources for *x*-, *y*-, and *z*- data in the graphs).



On the left side of the information bar is a drop-down menu that displays the symbolism and identifies the data source for each series currently linked. On the right side is an **Edit** button that opens the Data Source Properties dialog box in which you can set display names and data sources. Usually it is possible to unambiguously determine what data sources a graph has, but sometimes you need to indicate what data source to use, for example, when you plot a subrange of a data array. The information bar explains that you need to do this as soon as you turn on data linking; then, you can open the Data Source Properties dialog box to identify your data source(s).



Data Linking API

Use the `linkdata` function to turn data linking on or off for the current figure or for a figure for which you supply a handle.

Compatibility Considerations

If you capture figure windows with the `getframe` function, their images will include the message bar if the figures being captured possess them. To prevent this from happening, click the **X** on the right side of the message bar to dismiss it before calling `getframe`. Subsequent figures will not display a message bar. If you want to restore the message bar at a later time, select **Show linking and brushing message bar** in the MATLAB Preferences Confirmation Dialogs panel.

If the figure has a title, its Linked Plots/Data Brushing message bar can obscure it. This is the case for figures at the default size. Remove the message bar if you want a title to display.

Programming

Multithreaded Computations Enabled

Multithreaded computations, introduced in R2007a, are now on by default.

Compatibility Considerations

To disable multithreaded computations, open the Preferences dialog, choose Multithreading, and then disable it explicitly.

Enhancements to Object-Oriented Programming Capabilities

Major enhancements to object oriented programming capabilities enables easier development and maintenance of large applications and data structures.

New features include:

- The new `classdef` keyword enables you to define properties, methods, and events in a class definition file. See User-Defined Classes.
- A new `handle` class with reference behavior enables you to create more sophisticated data structures, such as linked lists and to manage external resources, such as files. See Comparing Handle and Value Classes.
- Events and listeners enable you to monitoring object property changes and other actions. See Events — Sending and Responding to Messages.
- Packages enable scoping of classes and functions. See Create a Namespace with Packages.
- Meta-classes provide support for class introspection. See Information from Class Metadata.
- JIT/Accelerator support provides significantly improved performance over the previous object oriented-programming system.

For a full description of object-oriented features, see Object-Oriented Programming.

Packages for Classes and Functions

This release provides the capability to manage name space by placing classes and functions in packages.

Clear Variables with Exceptions

With the new `clearvars` function, you can specify which variables you do not want cleared from memory.

Information on the State of Memory

The new `memory` function provides memory usage information such as largest block available, allowing you to diagnose memory problems on Microsoft Windows platforms.

Compatibility Considerations

The memory function existed in previous versions of MATLAB, but its purpose has changed. Previously, `memory` provided help text on how to free additional memory space for your MATLAB application. The function now returns information on the current state of memory use and availability in your system.

Define Your Own Function Cleanup Tasks

With the new `onCleanup` function, you can specify one or more tasks for MATLAB to perform just before exiting the current function.

New Functions

Name	Description
`clearvars`	Clear variables from memory
`memory`	Display memory information
`onCleanup`	Cleanup tasks at function completion

Extended JIT Support

JIT/Accelerator support now extends to statements executed at the MATLAB Command Line and in cell mode in the MATLAB Editor. This provides improved performance in these environments.

Enhancements to Image Information and Writing Functions

The image information and writing functions have the following enhancements:

- `imfinfo` can now return Exif data for JPEG or TIFF format image files. Information specific to the digital camera can be found in the `'DigitalCamera'` field, while any global positioning system information can be found in the `'GPSInfo'` field.
- `imwrite` now supports the `'RowsPerStrip'` parameter that you can use to specify how many image rows to include in a strip when writing TIFF files. By default, `imwrite` limits the number of rows included in a strip so that the size of the strip does not exceed 8 KB. Now you can specify strips of larger size.

Compression of -v7.3 MAT-Files

You can store data items that are over 2 gigabytes in size in a MAT-file using the `-v7.3` option for the `save` function. This option was introduced in MATLAB R2006b. With MATLAB R2008a, `save` now compresses these MAT-files.

Changes to Programming Documentation

Some of the chapters in the MATLAB Programming documentation have been moved or renamed in this release. Also the title of the Programming documentation has been changed to Programming

Fundamentals in order to differentiate this part of the MATLAB help from the new documentation on Developing MATLAB Classes.

Graphics and 3-D Visualization

New Figure Toolbar Buttons

Two new toolbar buttons and an information bar have been added in this release that control the new *Data Brushing* and *Data Linking* tools. Brushing and linking let you interactively explore and analyze data.

By default, now when you create axes or plot something into a blank figure, an informational banner appears across top of the figure with links to documentation for data brushing and linking capabilities. You can dismiss it by clicking the **X** button on right side of the message bar. Once you dismiss it, subsequent figures will not display the banner unless you select **Show linking and brushing message bar** in the MATLAB Preferences Confirmation Dialogs panel.

For more information, see “Data Brushing for Graphs and Linked Variables” on page 29-24 in the Data Analysis release notes and Interactive Data Exploration in the Data Analysis documentation. Also view the video tutorial that describes these and other new features.

“v6” Plotting Option Update — Affected Functions

The Version 7.5 (R2007b) release note “The “v6” Option for Creating Plot Objects is Obsolete” on page 30-29 identified plotting functions that accept the `v6` option, which is now obsolete and will be removed in a future version of MATLAB. There is no change to the status of these functions in R2008a. However, the list of affected functions in the R2007b release note had errors and omissions. Below is the correct list of functions that support the option in their syntax and now warn when it is used:

- `area`
- `bar`
- `barh`
- `colorbar`
- `contour`
- `contourf`
- `errorbar`
- `legend`
- `loglog`
- `mesh`
- `plot`
- `plot3`
- `quiver`
- `quiver3`
- `scatter`
- `scatter3`
- `semilogx`
- `semilogy`

- `stairs`
- `stem`
- `stem3`
- `subplot`
- `surf`

Note that the updated list adds functions `plot3` and `quiver3`.

In the earlier release note, the following functions were incorrectly identified as accepting the `v6` option:

- `meshc`
- `meshz`
- `surfc`

These functions do not call `v6` code and are not affected by it becoming obsolete.

Compatibility Considerations

Specifying the `v6` flag to any plotting function now results in a warning that the option is being removed, but the option still functions. To generate a FIG-file for a plot created with the `v6` option, you still need to use the `-v6` option to the `hgsave` command in order to save it in a form that a previous version of MATLAB can read. Figures containing annotations (such as textboxes, arrows, ovals, and rectangles) that are saved this way open in previous versions, but the annotations do not display because different objects are used to contain them in Version 7 than before. That is, the annotation objects have never been backward compatible.

Creating Graphical User Interfaces (GUIs)

New GUI Table Component

The new `uitable` component allows you to show data in a table. This component replaces the undocumented MATLAB `uitable` implementation. If you are using the old `uitable` component, please refer to the uitable Migration Document for help migrating to the supported `uitable` component.

Event Data Input to GUIDE Callbacks

Auto-generated callbacks of GUIDE GUIs can now access event data for Handle Graphics callbacks. The following Handle Graphics callbacks provide event data when triggered:

- `KeyPressFcn` in `uicontrol` and `figure`
- `KeyReleaseFcn` in `figure`
- `SelectionChangeFcn` in `uibuttonGroup`
- `WindowKeyPressFcn` in `figure`
- `WindowKeyReleaseFcn` in `figure`
- `WindowScrollWheelFcn` in `figure`
- `CellEditCallback` in `uitable`
- `CellSelectionCallback` in `uitable`

For example, the event data for keypress provides information on the key that is pressed. See the Callback Templates documentation for more information.

uigetfile and uiputfile Support of '.', '..', and '/'

Starting in R2007b, the `uigetfile` and `uiputfile` functions interpret `'.'`, `'..'`, and `'/'` the same way as does the `cd` command. `'.'` is interpreted as the current directory, `'..'` is the directory above the current directory, and `'/'` is the top level directory. When specifying a directory rather than a filename for either the `Filterspec` or `DefaultName` argument, you no longer need to end the string with a `'/'`. However, such strings ending with a `'/'` are interpreted as they were in previous releases.

hidegui Function Being Obsoleted

The hidegui function is being obsoleted and will be removed in a future version. Instead of this function use the `set` function to set the figure handle's `handlevisibility` property to `on` or `off`:

```
set(figurehandle, 'handlevisibility', 'on')
```

Changes to How uicontrols Set Figure SelectionType

`SelectionType` is a figure property that user interface components set when you click them. It is a read-only property that describes the gesture used when clicking the most recently selected object. For single mouse clicks, components no longer set the figure `SelectionType` property to anything but `'normal'` when they are enabled. By default, all components are enabled (their `Enable` property

is `'on'`). Previously, enabled components responded to modifier keys (**Ctrl**, **Alt**, **Shift**, and **Cmd**) by setting `SelectionType` to `'alt'` or `'extend'`. Now all key modifiers set `SelectionType` to `'normal'` for UI components.

Disabled components (those with their `Enable` property set to `'off'`) set `SelectionType` to `'alt'` or `'extend'` in response to modifier keys.

When `Enable` is `'on'`, the control is active and your `Callback` responds to clicks. When `Enable` is `'off'`, the control is not active, and your `ButtonDownFcn` responds to clicks. You can provide components with a `ButtonDownFcn` to detect changes in `SelectionType`.

An additional change affects how *list box* `uicontrol` components respond to double-clicking. The second of two successive clicks sets the `SelectionType` property of an enabled list box to `'open'`. Other types of uicontrols set their `SelectionType` to `'normal'` after both the first and the second clicks unless key modifiers were used.

The following table describes the click responses for `uicontrol` components.

Enable Status	Mouse Button Pressed	Key Modifier Pressed	SelectionType on First Click	SelectionType on Second Click
`on`	left		`'normal'`	`'normal'` *
`on`	right		`'alt'`	`'open'`
`on`	left	**Ctrl**	`'normal'`	`'normal'` *
`on`	right	**Ctrl**	`'alt'`	`'open'`
`on`	left	**Shift**	`'normal'`	`'normal'` *
`on`	right	**Shift**	`'extend'`	`'open'`
`off`	left		`'normal'`	`'open'`
`off`	right		`'alt'`	`'open'`
`off`	left	**Ctrl**	`'alt'`	`'open'`
`off`	right	**Ctrl**	`'alt'`	`'open'`
`off`	left	**Shift**	`'extend'`	`'open'`
`off`	right	**Shift**	`'extend'`	`'open'`

* Double-click result is `'open'` for list boxes.

For a two-button mouse, pressing the left and right buttons together (without key modifiers) can set `SelectionType` to either `'normal'` or `'extend'`; the results are unpredictable, possibly depending on which button you actually depressed first.

`uipushtool` and `uitoggletool` toolbar buttons do not set `SelectionType` at all, no matter what keys or mouse buttons you press. When they are disabled, `uipushtool` and `uitoggletool` controls do not execute their `ClickedCallback` (or `OnCallback`/`OffCallback`), nor does using them trigger a `WindowButtonDownFcn` callback.

Compatibility Considerations

If your callback code depends on the value for `SelectionType`, be aware that for the left mouse button, pressing **Ctrl** no longer sets it to `'alt'` and that **Shift**-clicking no longer sets it to `'extend'`.

The `SelectionType` behavior for the right mouse button has not changed.

External Interfaces/API

Interface to Generic DLLs Supported on 64-bit Platforms

The ability to load a generic DLL on 64-bit platforms using `loadlibrary` is available in MATLAB Version 7.6 (R2008a).

Compatibility Considerations

You must install a C compiler and Perl to use this feature. For a list of supported compilers and how to install them, see Using loadlibrary on 64-Bit Platforms.

Changes to Compiler Support

The set of compilers that MATLAB supports has changed in MATLAB Version 7.6 (R2008a). For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

New Compiler Support

MATLAB Version 7.6 (R2008a) supports new compilers for building MEX-files.

Microsoft Windows (64-bit) platform

- Microsoft Visual Studio 2008
- Windows SDK for Vista
- Intel Visual Fortran 10.1

Windows (32-bit) platform

- Microsoft Visual Studio 2008
- Open Watcom Version 1.7
- Intel Visual Fortran 10.1

Sun Solaris SPARC (64-bit) platform

- Sun Studio 12 cc / CC Version 5.9

Macintosh (Intel-based 32-bit) platforms

- Apple Xcode 3.0 (gcc / g++ Version 4.0.1)

Discontinued Compiler Support

The following compilers are no longer supported.

Windows platforms

- Intel C++ Version 7.1

- Intel Visual Fortran Version 9.0
- Borland® C++Builder® 6 Version 5.6
- Borland C++Builder 5 Version 5.5
- Borland C++ Compiler Version 5.5
- Compaq Visual Fortran Version 6.1
- Compaq Visual Fortran Version 6.6

Compatibility Considerations

To ensure continued support for building your C/C++ programs, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Compiler Support to Be Phased Out

The following compilers are supported in Version 7.6 (R2008a), but will not be supported in a future version of MATLAB.

Windows (32-bit) platform

- Open Watcom Version 1.3

Solaris SPARC (64-bit) platform

- Sun Studio 11 cc / CC Version 5.8

New Version of Perl on Windows Platforms

MATLAB Version 7.6 (R2008a) includes Perl on Windows Version 5.8.8.

Compatibility Considerations

Prior to this release, MATLAB contained Perl Version 5.005. Consult your Perl documentation for details on the changes between Perl versions.

Rebuild MEX-Files Created on Linux Platforms

MATLAB V7.6 (R2008a) on Linux platforms is built with a compiler that utilizes `glibc` Version 2.3.6.

Compatibility Considerations

To work with MATLAB V7.6 (R2008a), MEX-files compiled on a Linux platform must be rebuilt.

Use mxDestroyArray to Release Memory for mxArray

The documentation for the `mxSetCell`, `mxSetField`, and `mxSetFieldByNumber` functions in the MATLAB C and Fortran API incorrectly instructs customers to use `mxFree` to release memory for any `mxArray` returned by `mxGetCell`, `mxGetField`, or `mxGetFieldByNumber`.

Compatibility Considerations

The correct function to use is `mxDestroyArray`. Calling `mxFree` on an `mxArray` only frees the array header, but does not actually free the data itself and can result in a memory leak.

To help diagnose this problem, MATLAB issues a warning if calling `mxFree` on an `mxArray` could cause memory corruption. In future versions of MATLAB, this condition may result in a segmentation violation.

Do Not Use get or set Function to Manage Properties of Java Objects

If you want to read or update a property of a Sun Java object created in MATLAB using the Java class constructor, do not use the MATLAB `get` or `set` functions on the property. For example, if you create a Java object called `javaObject` that has a property called `PropertyName`, the following commands may cause memory leaks and will be deprecated in a future version of MATLAB:

```
propertyValue = get(javaObject, 'PropertyName');
set(javaObject, 'PropertyName', newValue);
```

Compatibility Considerations

The correct commands to use are:

```
propertyValue = javaObject.getPropertyName;
javaObject.setPropertyName(newValue);
```

In future versions of MATLAB, using `get` or `set` on Java objects to manage the properties will generate an error.

New mxArray Functions for Use with MATLAB Class Objects

You can read and modify properties of MATLAB class objects using the `mxGetProperty` and `mxSetProperty` functions.

mex.bat File Removed from matlabroot\bin\$ARCH

Beginning with MATLAB Version 7.3 (R2006b), the Windows script `mex.bat` is located in the directory *`matlabroot`*`\bin`. Copies of this file were also in the directory *`matlabroot`*`\bin\`*`$ARCH`*. In MATLAB Version 7.6 (R2008a), `mex.bat` is only located in *`matlabroot`*`\bin`.

Compatibility Considerations

If you did not make the updates described in “Location of mex.bat File Changed” on page 32-28, you may need to make changes now.

Run-time Libraries Required for Applications Built with Microsoft Visual Studio 2008 Compiler

If you distribute a MEX-file, an engine application, or a MAT-file application built with the Visual Studio 2008 compiler, you must provide the Visual C++ run-time libraries. These files are required to

run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed. For information on locating the Microsoft Visual C++ 2008 Redistributable Package (x86), containing `vcredist_x86.exe` and `vcredist_x64.exe`, consult your Microsoft documentation.

Environment Variables Required with Intel Visual Fortran 9.0

When you build a MEX-file, an engine application, or a MAT application using Intel Visual Fortran 9.0, MATLAB requires that you define an environment variable for the Windows platform you are using.

Windows (32-bit) platform

Define the environment variable `VS71COMNTOOLS`. The value of this environment variable is the path to the `Common7\Tools` directory of the Microsoft Visual Studio .NET 2002 or 2003 installation directory. (Intel Visual Fortran requires Visual Studio .NET 2002 or 2003 on 32-bit Windows platforms.) The Visual Studio .NET 2003 installation program commonly defines this environment variable. For example, you might set the environment variable as follows:

```
C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools
```

Windows x64 platform

Define the environment variable `MSSdk`. The value of this environment variable is the path to the installation directory for Microsoft Platform SDK for Windows Server® 2003. (Intel Visual Fortran requires Microsoft Platform SDK for Windows Server 2003 on Windows x64 platforms.) The Microsoft Platform SDK installation program does not commonly define this environment variable. For example, the environment variable might have the value

```
C:\Program Files\Microsoft Platform SDK
```

-largeArrayDims Option to MEX Will Become Default

In a future version of MATLAB, the default `mex` command will change to use the large-array-handling API. This means the `-largeArrayDims` option will be the default. For information about migrating your MEX-files to use the large-array-handling API, see the Technical Support solution `1-5C27B9`.

Compatibility Considerations

In the near future you will be required to update your code to utilize the new API. You should review your source MEX-files and `mex` build scripts.

Changes to Dynamic Data Exchange (DDE) Documentation

In MATLAB Version 5.1, all development work for the Dynamic Data Exchange (DDE) server and client was stopped. MathWorks provides, instead, a MATLAB interface to COM technology that is documented in Using COM Objects from MATLAB .

Obsolete Functionality No Longer Documented

Documentation for the following functions no longer included in External Interfaces.

Obsolete Functions
ddeadv
ddeexec
ddeinit
ddepoke
ddereq
ddeterm
ddeunadv

The following syntax for `enableservice` no longer included in External Interfaces.

```
enableservice('DDEServer',enable)
```

Compatibility Considerations

If you must support this obsolete functionality, we suggest you print and keep a copy of the relevant MATLAB function reference pages from V7.5 (R2007b) or earlier.

R2007b

Desktop Tools and Development Environment

Startup and Shutdown

Windows Platforms Startup Changes

You can now change the MATLAB startup directory on Microsoft Windows platforms using the standard shortcut **Start in** field. The `My Documents\MATLAB` subfolder (or `Documents\MATLAB` on the Microsoft Windows Vista platform) is the default startup directory. Upon startup, MATLAB automatically creates a `My Documents\MATLAB` subfolder (or `Documents\MATLAB` on the Windows Vista platform) if it does not exist, and adds it to the top of the MATLAB search path. To change the startup directory:

1. Right-click the MATLAB shortcut icon and select **Properties** from the context menu. The MATLAB Properties dialog box opens to the **Shortcut** pane.
2. In the **Start in** field, specify the directory in which you want MATLAB to start, for example, `C:\My MATLAB Place`.

   You can specify the directory via a UNC path (that is, the path can begin with `\\`).

The **Target** field specifies the full path to the file to start MATLAB, `matlab.exe`, located in the `bin` folder (for example, `C:\Program Files\MATLAB\R2007b\bin\matlab.exe`). Use the `bin\matlab.exe` to start MATLAB instead of `matlab.bat` or `matlab.exe` located in a platform directory such as `bin\win32`. The `bin\matlab.exe` detects the Windows platform and ensures required run-time files are installed.

Compatibility Considerations

The **Target** field no longer contains the `-sd $documents` startup option. In MATLAB Version 7.4 (R2007a), the startup directory was specified via the `-sd` startup option in the **Target** field. You had to specify the directory via a mapped drive. Any value in the **Start in** field was ignored.

The file to start MATLAB, as specified in the **Target** field, was `matlab.bat`.

Change any scripts you use to start MATLAB to specify the full path to `bin\matlab.exe`. If you use `matlab.bat` in R2007b, MATLAB issues a warning message instructing you to use `matlab.exe` instead.

If scripts include the `-sd` startup option to specify the startup directory, that will be the startup directory, even if a directory is specified in the **Start in** field.

Desktop

New features and changes introduced in Version 7.5 (R2007b) are:

- “Changes to Look of Buttons in Desktop and Other Tools” on page 30-4
- “Antialiasing Option No Longer Necessary on Windows and Macintosh Platforms” on page 30-5

Minimizing Tools in the Desktop Now Supported on Macintosh Platforms

You can now minimize tools in the desktop on Apple Macintosh platforms. It was introduced for other platforms in a previous version.

Double-Click to Maximize or Restore Minimized Tools in Desktop

After you minimize a tool within the desktop, you can now:

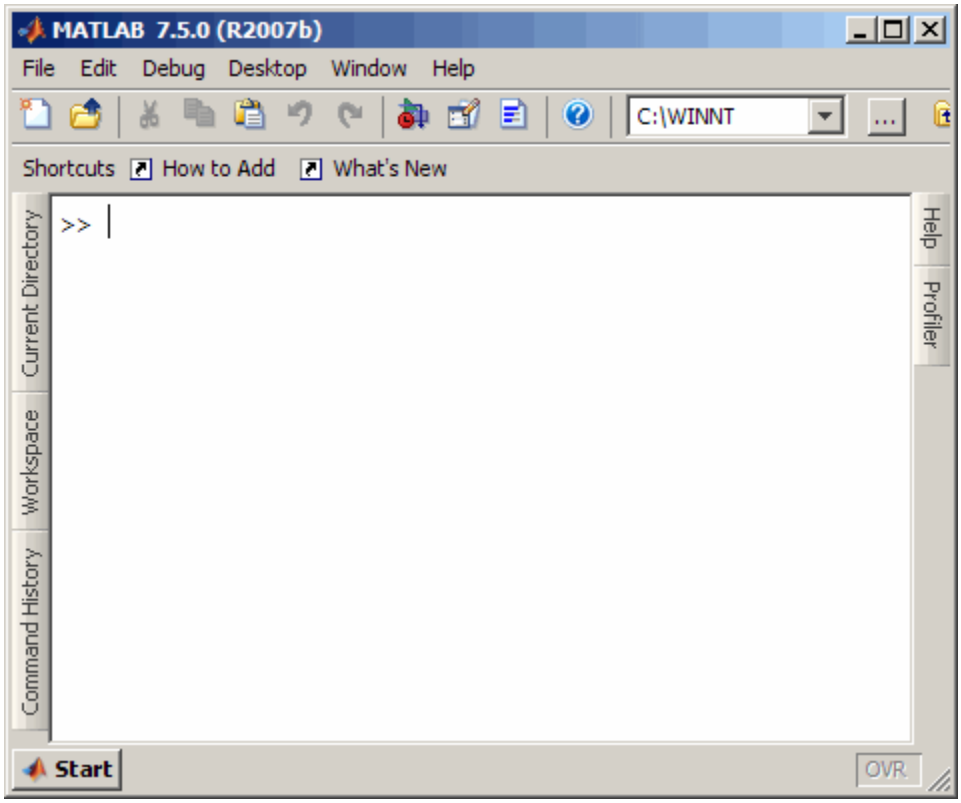
- Restore the tool to its former position by double-clicking the button.
- Drag a button to move its position—drag it to another edge of the desktop or to a new position within the edge where it's currently located.
- Restore the tool by dragging the button to a location within the desktop, or outside the desktop to undock the tool.

Similarly, you can double-click a tool's title bar to maximize the tool in the desktop; then double-click the title bar again to restore it to its former position. (The capability was introduced in R2007a, MATLAB Version 7.4).

For more information, see Open and Rearrange Desktop Tools and Documents.

New Desktop Layout — All but Command Window Minimized

Select **Desktop > Desktop Layout > All but Command Window Minimized** to arrange the desktop as shown here. The Command Window is open in the desktop, and all other desktop tools are open, but minimized.



Start Button Now Includes New Category for Links and Targets

In the Start button, there is a new category for Link and Target products. Select **Start > Links and Targets**, and then select one of the products. In previous versions, you accessed these products from the Toolboxes or Simulink software categories.

You also use this new category when running demos, or accessing **Demos** or **Contents** in the Help browser. For more information, see “Demos and Help Browser Contents Now Include New Category for Links and Targets” on page 30-5.

Start Button — View Source Files Renamed

To add your own toolboxes to the **Start** button, select **Start > Desktop Tools > View Start Button Configuration Files**. In previous versions, this menu item was **View Source Files**. There has been no change in functionality or features.

Changes to Look of Buttons in Desktop and Other Tools

Some icons on toolbar buttons have changed slightly. In addition, standard desktop icon image files are no longer provided in the *`matlabroot`*`/toolbox/matlab/icons` directory.

Compatibility Considerations

If your code relied on icon files in the *`matlabroot`*`/toolbox/matlab/icons` directory (for example, for adding your entries to the **Start** button or the Help browser), you might need to use other image files.

Antialiasing Option No Longer Necessary on Windows and Macintosh Platforms

MATLAB now follows the operating system's font settings on Microsoft and Macintosh platforms. This provides smooth fonts without the need for antialiasing within MATLAB.

Running Functions — Command Window and History

New features and changes introduced in Version 7.5 (R2007b) are:

Command History — Find Entry by Letter Now Looks in Collapsed Sessions

When you type letters in the Command History, it finds and selects the next entry that begins with the letters you typed. Now, if the entry is in a session that was collapsed, MATLAB automatically expands the session and selects the matching entry in it. In previous versions, MATLAB did not find matching entries in collapsed sessions.

If you do *not* want to find entries in collapsed sessions (the previous behavior), you can instead select **Edit > Find**, which finds text in the Command History, but not in collapsed sessions. For more information, see Quick Search for Entries Beginning with Specified Letters or Numbers.

Pop-Up Help for a Function in the Command Window

For more information, see “Help on Selection Enhanced in Command Window and Editor” on page 30-7.

Help

New features and changes introduced in Version 7.5 (R2007b) are:

Minor Visual Changes to Help Browser

- To open the Help browser from a tool's **Help** menu, select **Product Help**. In previous versions you selected **Full Product Family Help**.
- When the Help browser first opens, it displays help for MATLAB. In previous versions, it displayed a Begin Here page. The information previously available on the Begin Here page has been incorporated into the MATLAB roadmap page.
- When you close and reopen the Help browser, it maintains the list of pages you previously viewed, but does not open to the page you last viewed. In previous versions, upon reopening, the Help browser displayed the page you last viewed.

Demos and Help Browser Contents Now Include New Category for Links and Targets

When you run the `demo` function or access **Demos** or **Contents** in the Help browser, there is a new category for Link and Target products.

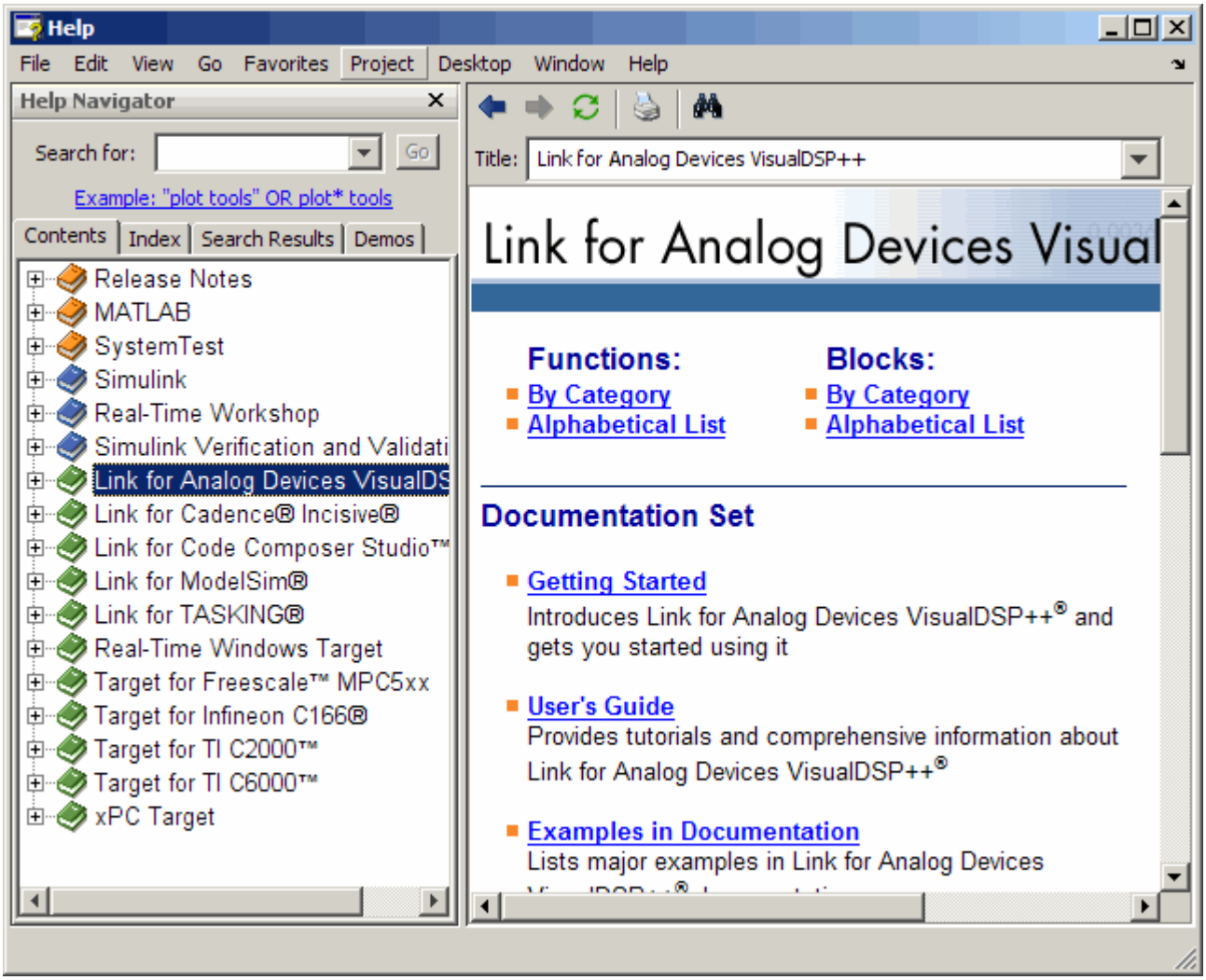
To use the `demo` function to access a demo that is now in the Links and Targets category, you specify the new subtopic 'links and targets', followed by the category. For example

```
demo('links and targets', 'link for modelsim')
```

displays the **Demos** pane, and expands the Link for ModelSim® demos listing.

In the Help **Demos**, Link and Target products appear together in their own category, and are identifiable by the new Links and Targets icon,  .

In the Help **Contents**, Link and Target products appear together after any installed Simulink and blockset products, and are identifiable by the new Links and Targets green book icon,  .



This new category is also used in the **Start** button in the MATLAB desktop—for more information, see “Start Button Now Includes New Category for Links and Targets” on page 30-4.

If you add help or demos to the Help browser for your own toolbox or list your own toolbox in the **Start** button and you want to take advantage of the new Links and Targets category, use the new type, `links_targets`, in the `info.xml` file for your toolbox.

Compatibility Considerations

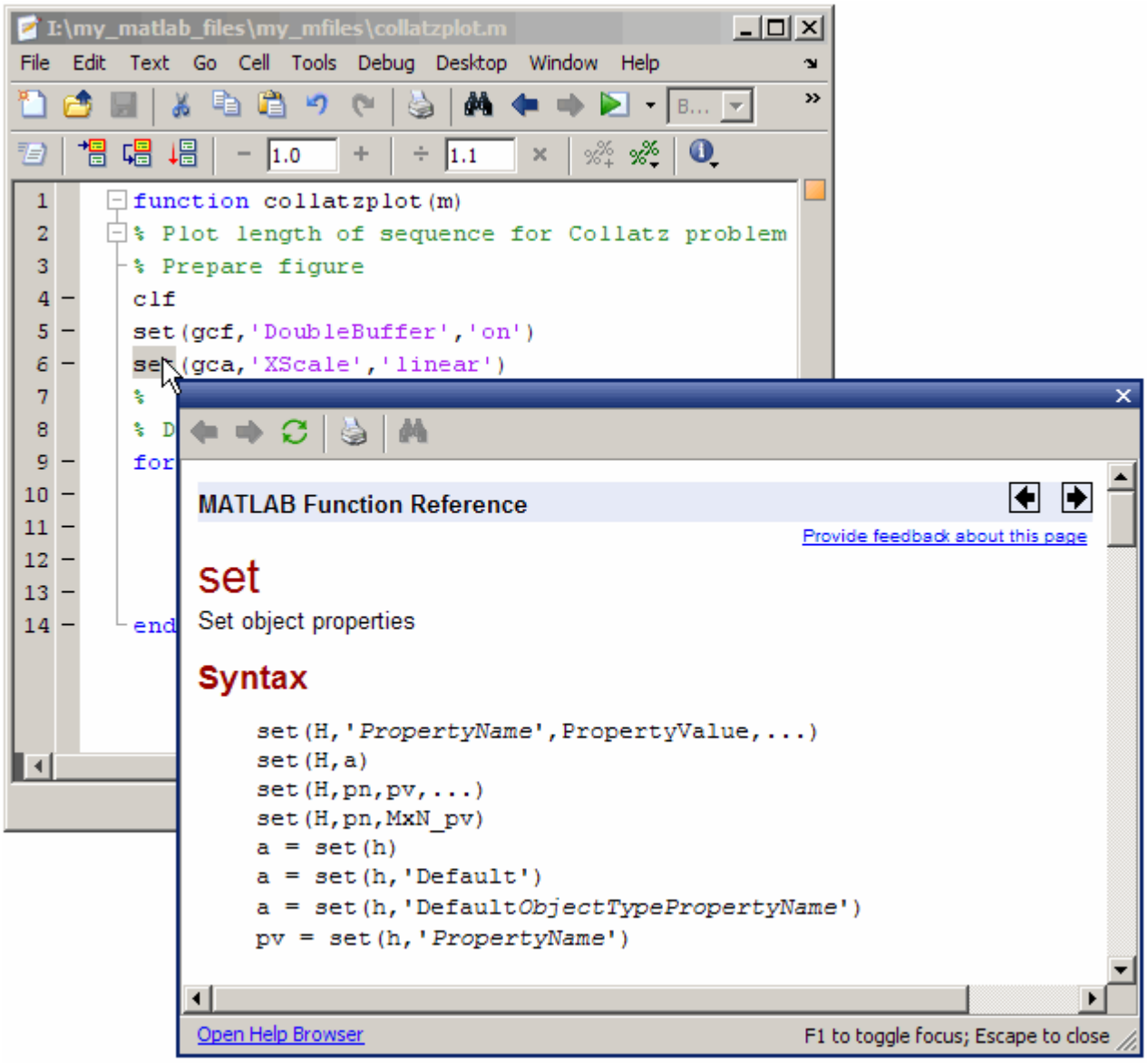
In previous versions, when you used the `demo` function to access a demo that is now in the Links and Targets category, you specified a different subtopic and category. If you have any code that relies on the `demo` function for accessing Links and Targets demos, you will need to replace the subtopic and category in the code.

In previous versions, you accessed the products in the Help browser **Demos** from the Toolbox or Simulink software categories.

In previous versions, you accessed the products in the Help browser **Contents** from within the list of toolbox products (orange book icon) or Simulink products (blue book icon).

Help on Selection Enhanced in Command Window and Editor

To get help for a function in the Command Window or the Editor, click the pointer in the function name and press **F1**. The reference page for that function appears in a small help window. To close the window, press **Escape**. You can also access the feature by choosing **Help on Selection** from the context menu. To change the window in which this help appears, select **File > Preferences > Help**, and adjust the option for **Help on Selection and More Help**.



Compatibility Considerations

In the previous version, you could select the function name, right-click, and select **Help on Selection**. The documentation appeared in the Help browser. Now if you want to see the documentation for the function in the Help browser, first access the pop-up help, and then click the **Open Help Browser** link.

Editing and Debugging M-Files

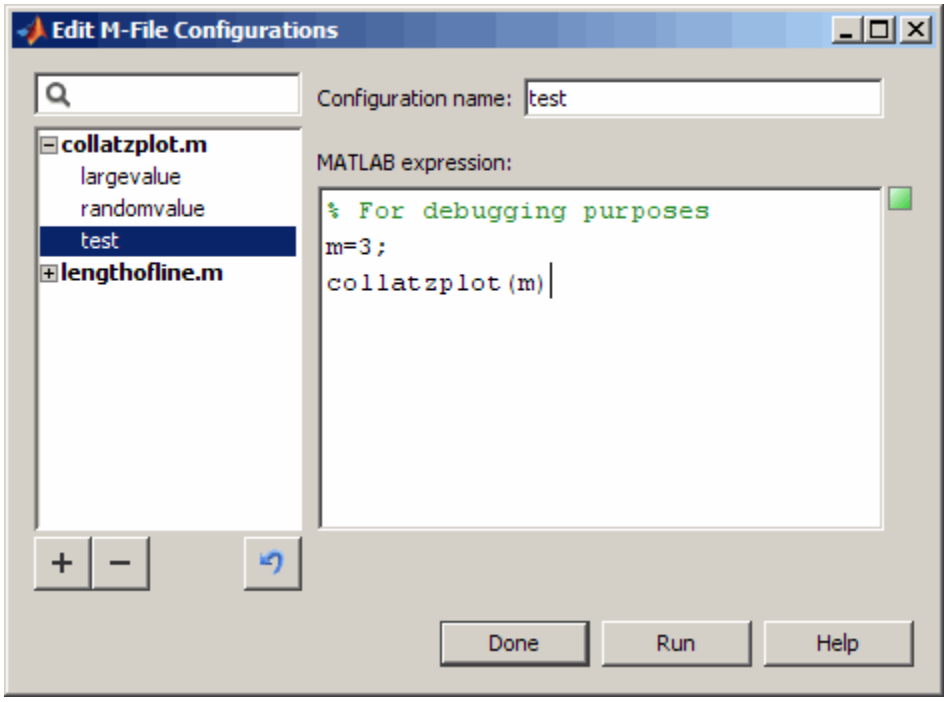
New features and changes introduced in Version 7.5 (R2007b) are:

- “Run Your Function M-Files in the Editor/Debugger Using Configurations” on page 30-9
- “Run/Continue Button Changes” on page 30-9

Run Your Function M-Files in the Editor/Debugger Using Configurations

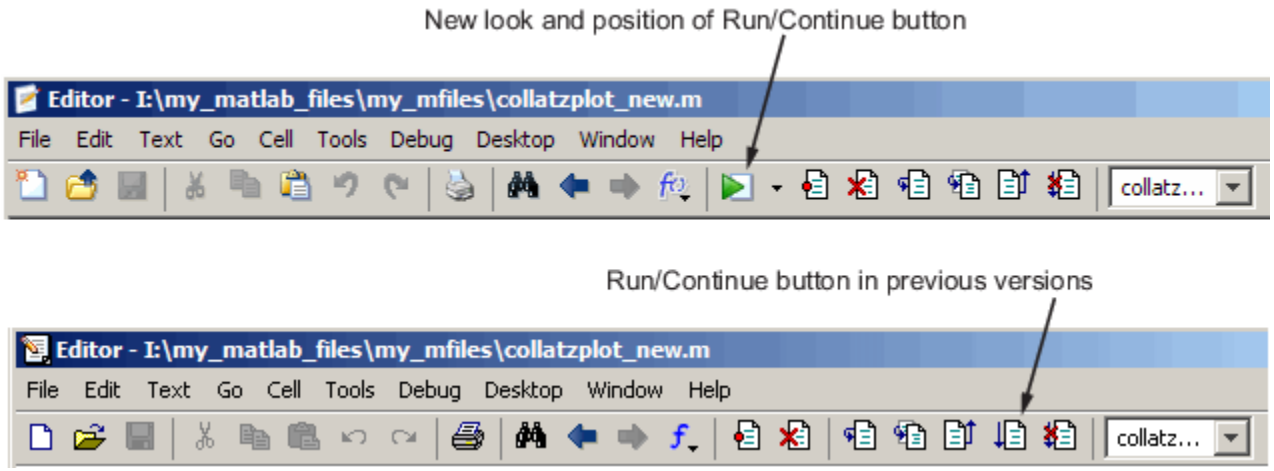
In the Editor/Debugger, you can provide values for a function's input arguments using a configuration, and then run that configuration to use the assigned values. Use a configuration as an alternative to running the function in the Command Window. You can associate multiple configurations with an M-file, each for different input values. MATLAB saves the configurations between sessions.

To create a configuration, first open an M-file in the Editor/Debugger. Then, from the down arrow on the Run button in the toolbar  select **Edit Configurations for *filename***. In the resulting Edit M-File Configurations dialog box, add statements and name the configuration. For more information, see Run Files with Input Arguments in the Editor.



Run/Continue Button Changes

The Run/Continue button has a new look and new location on the Editor/Debugger toolbar.

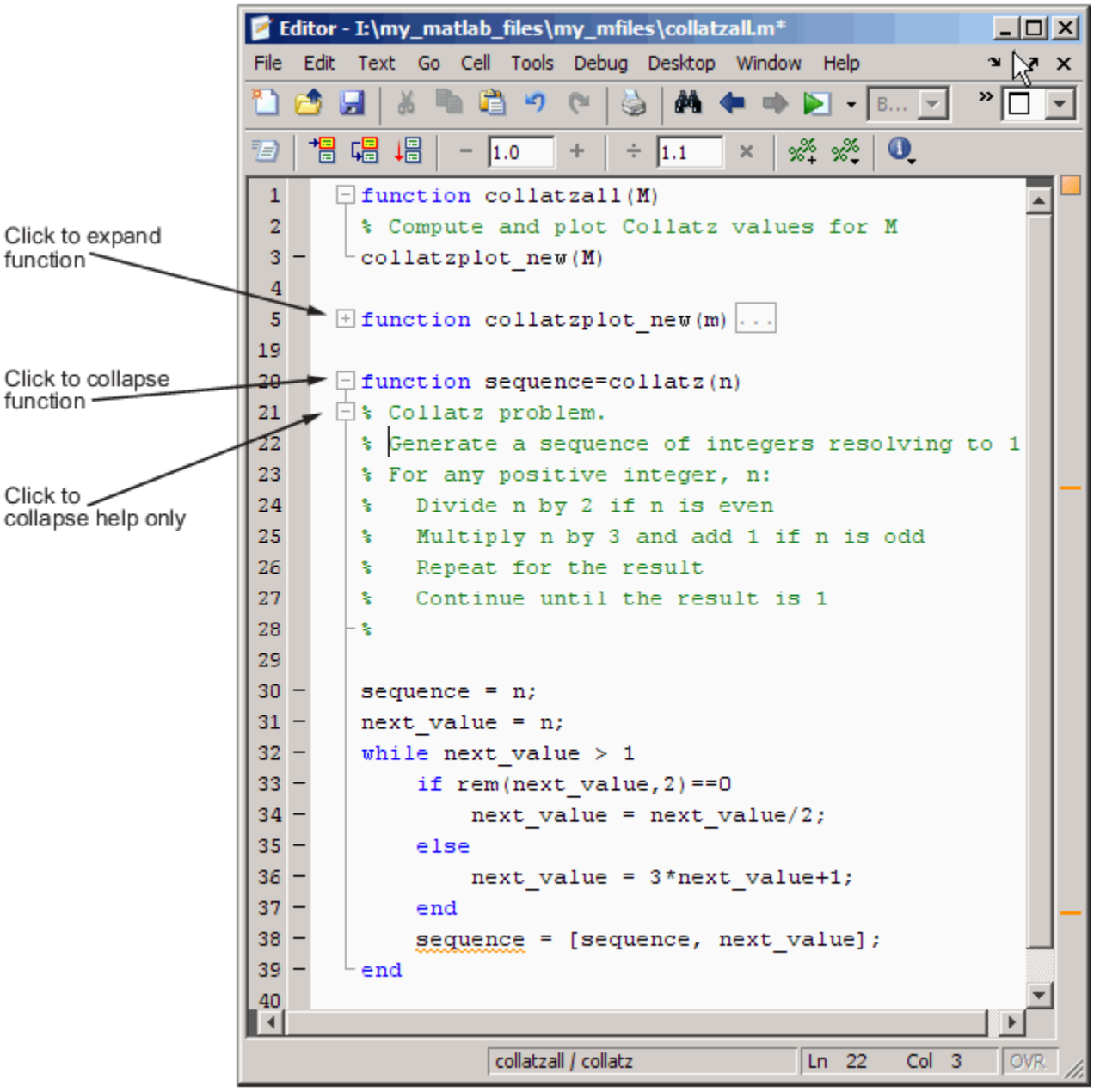


Compatibility Considerations

The button performs the same as it did in previous versions, but you need to access it in the new position.

Code Folding Feature for Collapsing and Expanding Code

To improve the readability of files containing several subfunctions, the Editor includes a code folding feature, which is enabled by default. Using this feature you can collapse and expand subfunctions and their associated help. The following figure shows the `collatzplot_new` function collapsed, such that only the function definition is displayed. The figure shows the `collatz` function expanded, revealing both the help code and the function code. If you collapse just the help code, only the H1 help line displays.



- To expand code that is collapsed, click the plus sign (+) to the left of the code you want to expand.
- To collapse code that is expanded, click the minus sign (-) to the left of the code you want to collapse.
- To expand or collapse all of the code in an M-file, place your cursor anywhere within the M-file, right-click, and then select **Code Folding > Expand All** or **Code Folding > Collapse All** from the context menu.

For more information, see Code Folding—Expanding and Collapsing M-File Constructs

Quick Help for a Function in the Editor

For more information, see “Help on Selection Enhanced in Command Window and Editor” on page 30-7.

Line Endings Removed in Files Provided with MATLAB Software for Windows Platforms; Impacts Viewing in Notepad Application

In previous versions, text files provided with MATLAB for Windows platforms included a carriage return and line feed at the end of each line. Starting in R2007b, the text files MATLAB provides do not include a carriage return and line feed at the end of each line.

File types affected are:

- `.asc`
- `.bat`
- `.c`
- `.cc`
- `.cdr`
- `.cpp`
- `.def`
- `.for`
- `gs.rights`
- `.h`
- `.ini`
- `.m`
- `.mdl`
- `.pl`
- `readme`
- `.tlc`
- `.tmf`
- `.txt`

There is no impact if you view the files in MATLAB and other common text editors, with the known exception of the Microsoft Notepad application.

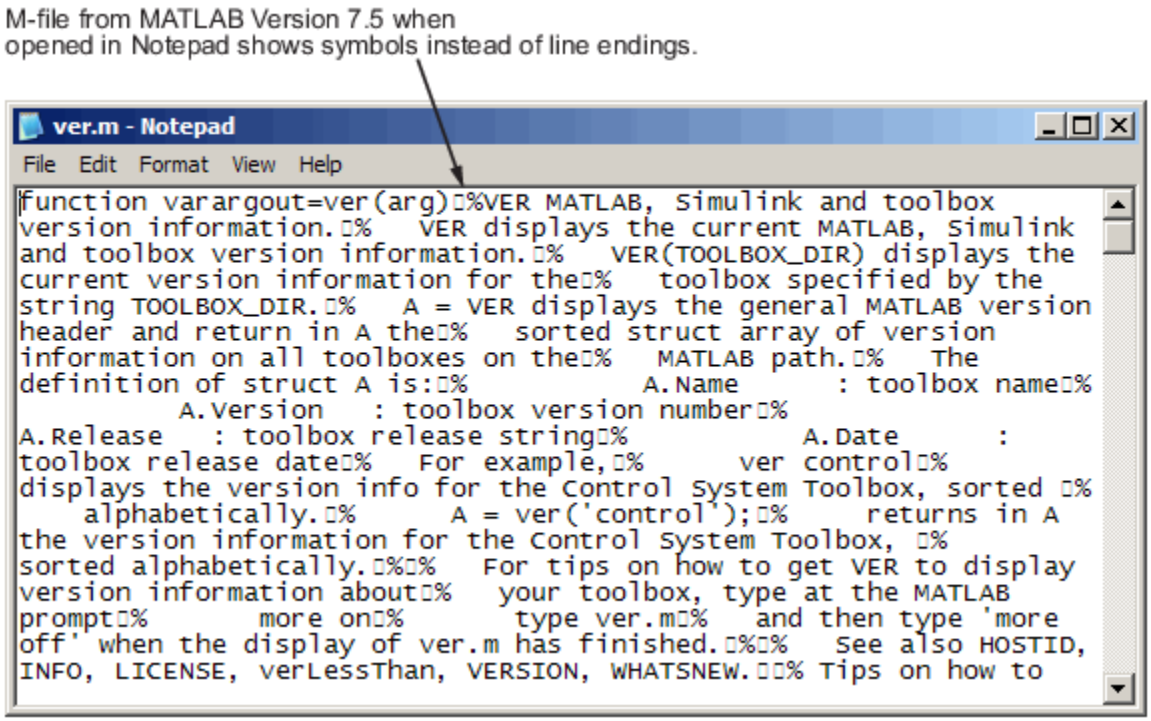
Compatibility Considerations

If you use the Notepad application to view files provided with MATLAB, you see carriage return and line feed symbols  instead of line endings. This makes the files less readable in the Notepad application. Other text editors might display the symbols instead of line endings, but of the common text editors tested, none have been found that do so.

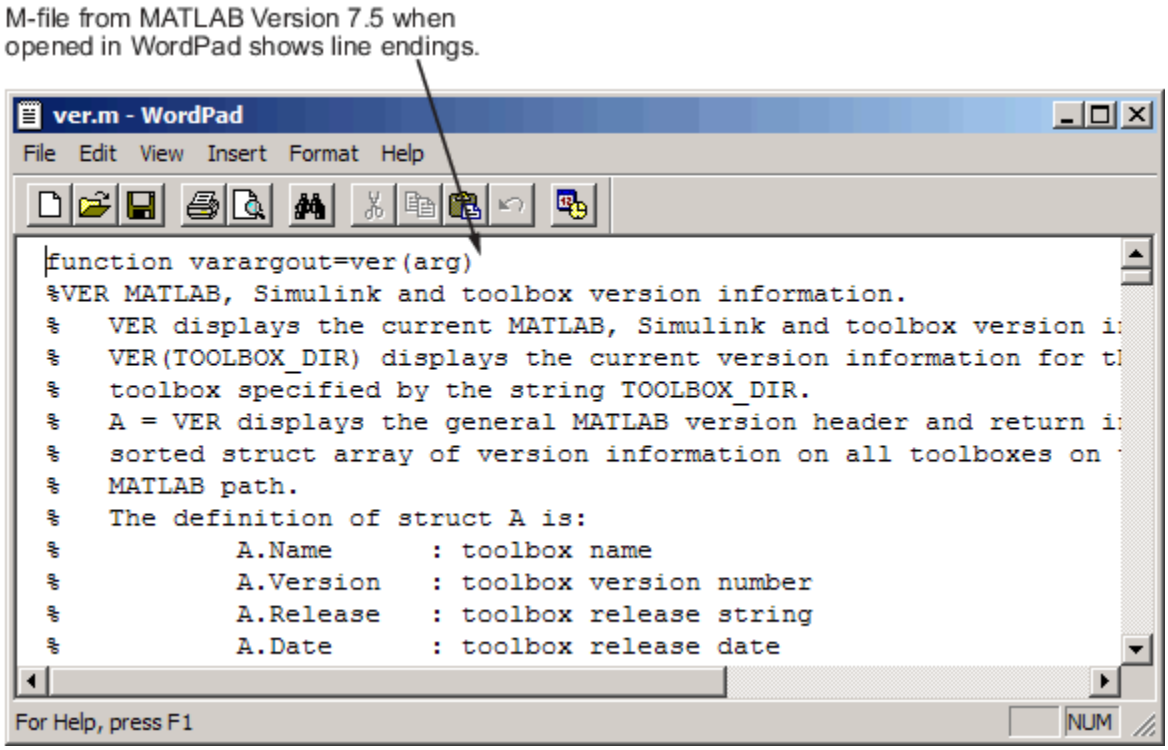
As an alternative to the Notepad application, use the Microsoft WordPad application, provided with Windows platforms, or another text editor to view the files.

If your Windows file associations are set to associate any of the listed file types with Notepad, change the associations to use WordPad or another text editor.

The following illustration shows how the `ver` M-file included with MATLAB Version 7.5 looks when opened in the Notepad application.



The following illustration shows how the `ver` M-file included with MATLAB Version 7.5 looks when opened in the WordPad application.



There are no problems with files you create or edit in the Notepad application, and then view or edit in MATLAB. The files have line endings in the MATLAB Editor, and continue to have line endings when you open them in the Notepad application.

(Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.)

Stand-Alone Editor Will Not Be Included in Next Version

The MATLAB stand-alone Editor (`meditor.exe`) will no longer be provided, starting in the next version of MATLAB. Instead of the stand-alone Editor, you can use the MATLAB Editor/Debugger. It provides all the features of the stand-alone Editor, plus the following:

- Tab completion
- Debugging M-files
- Evaluating selections
- Accessing source control features
- Docking the tool in the MATLAB desktop
- Using cell features for rapid code iteration or publishing

Compatibility Considerations

Some users have preferred the stand-alone Editor to the MATLAB Editor/Debugger because of slightly better startup performance and because it does not require a MATLAB license. For those situations, you can use any text editor you have, such as the UltraEdit® application from IDM Computer Solutions, or the GNU Emacs software.

Determine the McCabe (Cyclomatic) Complexity of an M-File

The `cyc` option to the mlint function enables you to determine the McCabe complexity (also referred to as the cyclomatic complexity) of an M-file. Higher McCabe complexity values indicate higher complexity, and there is some evidence to suggest that programs with higher complexity values are more likely to contain errors. Frequently, you can lower the complexity of a function by dividing it into smaller, simpler functions. In general, smaller complexity values indicate programs that are easier to understand and modify. Some people advocate splitting up programs that have a complexity rating over 10. See `mlint` for syntax and an example.

Publishing Results

New features and changes introduced in Version 7.5 (R2007b) are:

Notebook and Word for Office 2007

Notebook now supports Microsoft Word for Office 2007. For details, see Creating a MATLAB Notebook to Publish to Microsoft Word.

Text Markup in Cells for Publishing

The following Editor/Debugger menu items are added to assist you in marking up cells in the Editor/Debugger for publishing. Access the menu items presented in the following list from **Cell > Insert Text Markup** . When you select the menu item, the Editor inserts code to assist you in adding the text markup for the specified item.

- **Document Title and Introduction**
- **Section Title with Cell Break**
- **Hyperlinked Text**
- **Image**
- **Numbered List**
- **HTML Markup**
- **LaTeX Markup**

The first two list items are provided instead of the **Cell Title** and **Descriptive Text** menu items that were offered in Version 7.4 (R2007a).

As an alternative to using the Cell menu items, you can manually insert code to mark up cells in your M-file for publishing. For details on the Cell menu items and the resulting code see Marking Up Text in Cells for Publishing

Preference to Restrict Lines of Output

You can now specify options to restrict the number of lines included in the output of a published M-file. To access this option from the Editor/Debugger, follow these steps:

1 Select **File > Preferences > Editor/Debugger > Publishing**

2. In the Editor/Debugger Publishing Preferences pane, select the **Evaluate code** and **Restrict output to** options.
3. Specify the maximum number of lines that you want to include in the output.

Mathematics

New Functions

Function	Description
`quadgk`	Numerically evaluates the integral, adaptive Gauss-Kronrod quadrature
`bvp5c`	Solves boundary value problems for ordinary differential equations, notably useful for small error tolerances
`maxNumCompThreads`	Gets and sets the maximum number of computational threads

finite Function Deprecated

In this release, the `finite` function displays a warning message that the function is now deprecated. Support for `finite` will be removed in a future release of MATLAB software.

Compatibility Considerations

It is recommended that you replace all calls to `finite` with the `isfinite` function.

dmperm Function Gives Coarse Decomposition

The `dmperm` function now provides two additional output arguments for the indices of the Dulmage-Mendelsohn coarse decomposition.

ldl Function Supports Real Sparse Symmetric Matrices

The `ldl` function now provides factorization and solving for an additional output argument, the scaling matrix, when the input matrix is real sparse and symmetric.

Upgrade to LAPACK Library

MATLAB software now uses Version 3.1 of the Linear Algebra Package (LAPACK) library.

Upgrade to BLAS Libraries

For AMD processors, MATLAB software now uses Version 3.6 of the AMD Core Math Library (ACML™) for the Basic Linear Algebra Subroutine (BLAS) libraries.

Library for LAPACK and BLAS Symbols Separated

The binder library, `libmwlapack.lib`, containing both LAPACK and BLAS symbols is now two separate library files: `libmwlapack.lib` for LAPACK symbols and `libmwblas.lib` for BLAS symbols.

Compatibility Considerations

If you previously linked to the `libmwlapack.lib` library to use the BLAS symbols, you will need to update your code to link to the `libmwblas.lib` library.

Colon Operations on Characters Return Character Type Data

Using a colon with characters to iterate a for-loop now returns data of type character. For example,

```
for x='a':'b',x,end
```

results in

```
x =
    a
x =
    b
```

Compatibility Considerations

Previously, colon operations with characters iterating a for-loop returned data of type double. In previous releases the above example returned:

```
for x='a':'b',x,end

x =
    97
x =
    98
```

Existing program code that relies on the colon operations of character arrays returning a double, needs to be updated to expect a character data type.

Matrix Generating Functions No Longer Accept Complex Inputs

Calling matrix generating functions, such as `ones`, `zeros`, `rand`, `randn`, `true`, and `false`, with a complex number as dimensions input now returns the error:

```
true([1 i])
??? Error using ==> true
Size vector must be a row vector with real elements.
```

Compatibility Considerations

In previous releases, if you supplied a complex number as a dimension input, MATLAB software returned:

```
true([1 i])
Warning: Size vector should be a row vector with integer elements.
         Complex inputs will cause an error in a future release.

ans =

   Empty matrix: 1-by-0
```

Existing program code that relies on entering complex numbers as dimension input to a matrix generating function should be modified.

Data Analysis

Programming

Increased Size for Large Arrays

On 64-bit platforms, MATLAB arrays are no longer limited to 2^{31} elements. The limit in MATLAB 7.5 is 2^{48}-1. For example, given sufficient memory, many numeric and low-level file I/O functions now support real double arrays greater than 16 GB.

Documentation for Multiprocessing in MATLAB

Documentation for “Multiprocessing in MATLAB” has moved from Desktop Tools and Development Environment to the “Improving Performance and Memory Usage” section of MATLAB Programming.

Setting Number of Threads Programmatically

In this release, MATLAB provides a way to set or retrieve the maximum number of computational threads from within an M-file program. With the `maxNumCompThreads` function, you can either set the maximum number of computational threads to a specific number, or indicate that you want the setting to be done automatically by MATLAB

New Internal Format for P-code

P-code files have a new internal format in MATLAB Version 7.5. The new P-code files are smaller and more secure than those built with MATLAB 7.4 and earlier, and provide a more robust solution to protect your intellectual property.

Any P-code files that were built using MATLAB 7.4 or earlier also work in 7.5. However, support for these older files will at some point be removed from MATLAB.

P-code files built with MATLAB 7.5 only work on 7.5 or later. They cannot be used with MATLAB 7.4 or earlier versions.

Compatibility Considerations

Rebuild any P-code files using MATLAB 7.5 that you expect to need in the future.

New Split String Functionality in regexp

Using the regular expressions function `regexp`, you can now split an input string into sections by specifying the new `'split'` option when calling `regexp`:

```
s1 = ['Use REGEXP to split ^this string into ' ...
      'several ^individual pieces'];

s2 = regexp(s1, '\^', 'split');

s2(:)
ans =
    'Use REGEXP to split '
    'this string into several '
    'individual pieces'
```

The `split` option returns those parts of the input string that are delimited by those substrings returned when using the `regexp 'match'` option.

Changes Related to Error Handling

New Error Handling Mechanism

MATLAB extends its error-handling capabilities with the new `MException` class to provide you with a more secure and extensible system for throwing and responding to errors. Read about this new feature in the Error Handling section of the MATLAB Programming documentation and in related function reference pages such as `MException`.

This feature *extends* the error-handling capabilities available in earlier releases, but does not replace that functionality. Your M-file programs will continue to function the same when using Version 7.5.

New Syntax for catch Function

As part of new error-handling mechanism, the `catch` function has a new, optional syntax as shown here in a try-catch statement:

```
try
   % Try to execute this block of code. Go to 'catch' on error
   - code that may error -
catch ME
   % Deal with the error captured in MException object, ME
   - code to handle the error -
end
```

`ME` is an object of the `MException` class. This command gives you access to the `MException` object that represents the error (i.e., exception) being caught.

Warning and Error Messages Now Wrap

In previous versions of MATLAB, warning and error messages that were longer than the width of your terminal screen extended beyond the visible portion of your screen. These messages now wrap onto succeeding lines so that the entire text of the warning or error is visible.

Change to Error Message from Anonymous Function

The error message and M-file line number that MATLAB displays when encountering an error in an anonymous function defined within a script file or at the MATLAB Command Line has changed in this release. In MATLAB Version 7.4, the error message included the line number (set to `1` for anonymous functions), and the stack information returned by the `lasterror` function also showed the line number as `1`. In MATLAB 7.5, the line number is not displayed in the error message and is set to `0` in the returned stack information.

For example, when you enter the following two lines at the command line, MATLAB generates an error from the anonymous function:

```
X = @() error('* Error *');
X()
```

This example shows the difference between MATLAB Versions 7.4 and 7.5:

```
e = lasterror;
```

```
e.message
ans =
Error using ==> @()error('Error') at 1    % V7.4 response
Error using ==> @()error('Error')         % V7.5 response
* Error *

e.stack
ans =
    file: ''
    name: '@()error('* Error *')'
    line: 1                               % V7.4 response
    line: 0                               % V7.5 response
```

Compatibility Considerations

If you have programs that rely on the line number returned in response to an error in an anonymous function, these programs may not work as expected. Remove dependencies on the returned error message and line number, or update your program code to use the new string and value.

New Message In Response to Ctrl+C

MATLAB now displays a more user-friendly message when you press **Ctrl+C**. The previous response to **Ctrl+C** was

```
Error in ==> testctrlc>waitawhile at 5
pause(100);

Error in ==> testctrlc at 2
waitawhile
```

In this and future releases, pressing **Ctrl+C** still halts program execution, but now displays the response

```
??? Operation terminated by user during ==> testctrlc>
   waitawhile at 5

In ==> testctrlc at 2
waitawhile
```

Compatibility Considerations

You only need to be aware that the change in the text of this message is intentional and does not signify any error on your part.

hdfread Errors Instead of Warns on I/O Failures

In previous releases, `hdfread` issued a warning when a requested I/O operation failed. In addition, `hdfread` created an empty variable in the workspace. In this release, `hdfread` now errors when a requested I/O operation fails and does not create an empty variable in the workspace.

Compatibility Considerations

If you call `hdfread` in a script to perform an I/O operation and that operation fails, your script will now terminate. Previously, because `hdfread` only warned when an I/O operation failed, your script would continue processing.

Results From tempname Are More Unique

The `tempname` function now produces a string such as

```
C:\Temp\tpe51f2ba3_9ad3_490f_8142_58359c98f4a5
```

when Java is present, or a string like

```
C:\Temp\tp346976948758473
```

when `nojvm` is selected. Underscores are included in the name so you can use the filename portion of it as a valid M-function name. If a string row vector is passed in as an argument, that string is used instead of `tempdir` as the root.

Compatibility Considerations

Because the new string generated by `tempname` is generally longer than the string constructed in earlier versions, there is a possibility of exceeding length restrictions, especially if your program code passes a string to the `tempname` function. If you consider this to be a potential problem, verify that the strings you pass to `tempname` will not result in an overly long string being returned.

MATLAB Includes New Input Argument Validation Functions

MATLAB now includes two new functions that validate the input arguments passed to functions. For example, you can use these functions to make sure that an input argument is numeric and nonempty. The following table lists these functions with a brief description.

Function	Description
`validateattributes`	Check validity of array
`validatestring`	Check validity of text string

Windows Current Working Directory Corrected

On Windows, you can define a current working directory, `cwd`, for each drive letter. For example, entering the command `cd D:\work` at the DOS prompt defines your `D` current working directory as `D:\work`. All references to `D:` are then relative to this directory.

The term . . .	Represents the directory . . .
`D:`	`D:\work`
`D:matlab`	`D:\work\matlab`
`D:\matlab`	`D:\matlab`

(Note the difference between `D:\` and `D:`, where the former is the drive `D`, and the latter is a user-defined working directory that may or may not be equal to `D:\`.)

Previous versions of MATLAB have been inconsistent in the way that volume-relative path specification is handled. For example, in MATLAB 7.4 and earlier, if `D:` were defined as the directory `D:\work`, the following commands on the left and right returned identical results:

```
dir D:                              dir D:\
dir D:matlab                        dir D:\work\matlab
fileparts('D:myfile.m')             fileparts('D:\myfile.m')
```

This has been fixed in MATLAB 7.5 so that the following are now equivalent:

```
dir D:                              dir D:\work
dir D:matlab                        dir D:\work\matlab
fileparts('D:myfile.m')             fileparts('D:\work\myfile.m')
```

Compatibility Considerations

Some MATLAB commands may fail or return unexpected results if you use Windows current working directories on MATLAB. You might have to update hard-coded paths if you have been relying on the incorrect behavior exhibited in earlier versions.

New Multimedia Functionality

A new `mmreader` video file reader object for Windows platforms supports formats such as AVI, MPEG, and WMV, and adds the ability to read additional video codecs that `aviread` does not support. For more information, see the `mmreader` and `read` reference pages.

Compressed AVI Video Files in Windows Vista and Windows XP x64

Because Windows Vista, Windows Vista 64-bit, and Windows XP x64 operating systems do not ship with the Indeo® 5 codec, which is the default codec used by MATLAB Audio-Video functions for file compression, the functions `movie2avi` and `avifile` now generate uncompressed AVI files on these platforms. Also, `aviread` on these platforms cannot read files that were compressed using the Indeo 5 codec.

Compatibility Considerations

If you have upgraded your Windows operating system to Windows Vista, Windows Vista 64-bit, or Windows XP x64, you need to install the Indeo 5 codec to read or create Indeo 5 compressed AVI files with `aviread`, `avifile`, or `movie2avi`. You can learn more about downloading the Indeo 5 codec from the Ligos Corporation Web site at `http://ligos.com/index.php/home/products/indeo/`

mmfileinfo Reads Files on MATLAB Path

The `mmfileinfo` function now reads files on the MATLAB path, not only those in the current directory. The `mmfileinfo` output struct contains a field called `Path`, which indicates the directory where the file exists.

Compatibility Considerations

The `Filename` field of the output struct from `mmfileinfo` now contains only the filename itself without any path information, while the path information is contained in the `Path` field. In previous releases, the `Filename` field contained both path and filename information.

Changes to imread Support of TIFF Format

The `imread` function includes several updates to its TIFF support:

- `imread` reads TIFF files that use JPEG, LZW, and Deflate compression.
- `imread` reads image data from TIFF files in any arbitrary samples-per-pixel and bits-per-sample combination.
- `imread` provides increased performance when reading large images, when used with the 'PixelRegion' parameter.

Removal of freeserial Function

The `freeserial` function is now obsolete. Use `fclose` to release the serial port.

Compatibility Considerations

If your program code still makes use of the `freeserial` function, replace each instance with the `fclose` function instead.

Graphics and 3-D Visualization

Datatips Are Now Saved to FIG-Files

When you save a figure, all datatips existing in it are saved along with other annotations. When you open the FIG-file, the datatips are displayed and can be manipulated or deleted in the same ways they could in the original figure.

Compatibility Considerations

If you open a FIG-file containing datatips while using a previous MATLAB version (V7.4 or earlier), no error results, but the datatips do not display.

New Options for Displaying Groups of Lines in Legends

You can now customize how legends for figures display groups of lines, such as contours. Previously, legends displayed groups of lines such as contourgroups with a glyph that represented the entire group; now users have the flexibility to designate a single legend entry, a legend entry for each child of the group, or no legend entries for the group.

By default, a legend entry for an hggroup now consists of the `DisplayName` of its first child and a glyph representing it (previously, no glyph appeared, only the `DisplayName`). This is what you now see after clicking the legend tool icon in the figure's toolbar. However, you can set the new `Annotation` property of hggroups to control how the group is represented in a legend. For details and examples of its use in customizing legends, see Controlling Legends in the Graphics documentation.

Drawnow Update Option Now Updates Uicontrols Only

The `drawnow` command can now selectively update the display of UI components. The `update` option enables you to update only uicontrol objects without allowing callbacks to execute or processing other events in the queue.

Annotation Textboxes Can Automatically Resize to Fit their Contents

In previous releases, textboxes had fixed sizes that users needed to adjust to fit the size of their contents. Now, if you create a textbox annotation using a GUI or the `annotation` function, it can grow or shrink to just fit the text you type into it. This behavior is controlled by the Annotation Textbox object's `FitBoxToText` property, which can be `'on'` or `'off'`. When you create a textbox with the Annotation toolbar (using the  tool), this property is set to `'on'` if you create a textbox without dragging; however, if you drag to make the new textbox have a certain size, the property is initially `'off'`. When you create a textbox with the annotation function, for example,

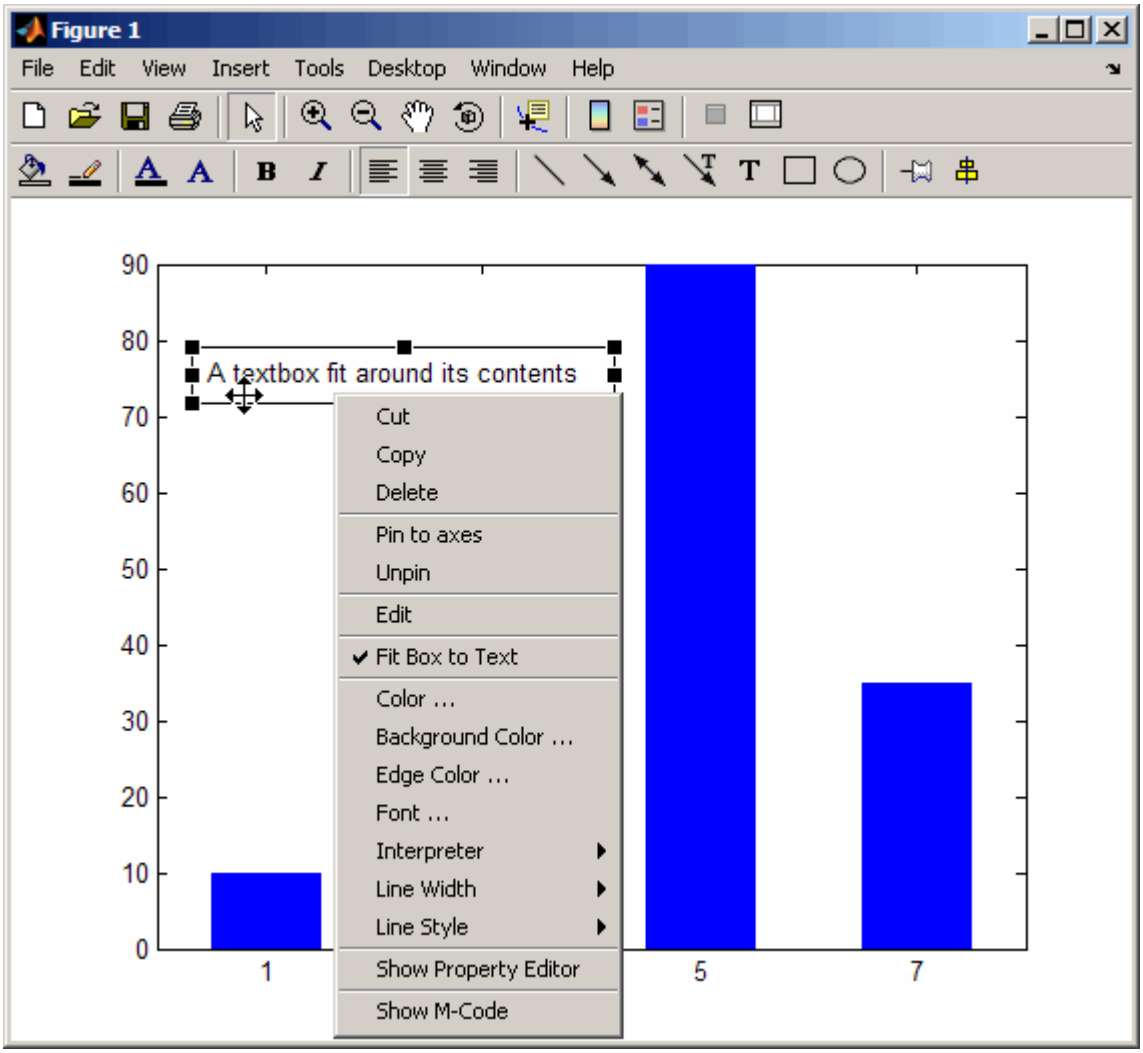
```
htb = annotation('textbox')
```

without specifying a position and size, the textbox is created with `FitBoxToText` set to `'on'`. If you specify a position vector in the command, for example,

```
htb = annotation('textbox', [.1 .8 .4 .1])
```

the textbox is created with `FitBoxToText` set to `'off'`.

Similarly, if you resize a textbox in plot edit mode or change the width or height of its `position` property directly, its `FitBoxToText` property is set to `'off'`. You can toggle this property with `set`, with the Property Inspector, or more conveniently, via the object's context menu, as the illustration below shows.



If you edit a textbox that has `FitBoxToText` set to `'on'`, the textbox resizes to accommodate the number of characters and lines in the text as you type. You can reposition the textbox without changing the `FitBoxToText` property, but as soon as you resize it, the property becomes (or remains) `'off'`.

Property Inspector Now Has Context-Sensitive Help

When you use the Property Inspector (the `inspect` command), you can now ask for a description of any property it shows. The descriptions come from the property reference page for the type of object being inspected (e.g., axes, lineseries, annotations, uicontrols, etc.). To get a description of a property, right-click its name in the left-hand column of the Property Inspector and select **What's This?** from the context menu that appears. A mini-help window opens to show the property's description from the object's property reference page. If you need an overview of all properties

pertaining to particular kinds of objects and how these objects relate, scroll through the entries in the mini-help window or open the Handle Graphics Property Browser from the Help Browser.

The “v6” Option for Creating Plot Objects is Obsolete

Prior to MATLAB Version 7, handles returned by high-level plotting functions referenced graphic primitives, such as lines and patches. In Version 7, MATLAB began bundling the primitives into “series” objects, for example, lineseries, barseries, and contourgroups. At that time, the `v6` option was added to plotting functions to enable users of MATLAB Version 7.x to create FIG-files that previous versions can open. The `v6` option is now obsolete. It will be removed in a future MATLAB release. The following functions include the option in their syntax and now warn when it is used:

- `area`
- `bar`
- `barh`
- `colorbar`
- `contour`
- `contourf`
- `errorbar`
- `legend`
- `loglog`
- `mesh`
- `meshc`
- `meshz`
- `plot`
- `quiver`
- `scatter`
- `scatter3`
- `semilogx`
- `semilogy`
- `stairs`
- `stem`
- `stem3`
- `subplot`
- `surf`
- `surfc`

Compatibility Considerations

Specifying the `v6` flag to any plotting function now results in a warning that the option is being removed, but the option still functions. To generate a FIG-file for a plot created with the `v6` option, you still need to use the `-v6` option to the `hgsave` command in order to save it in a form that a previous version of MATLAB can read. Figures containing annotations (such as textboxes, arrows, ovals, and rectangles) that are saved this way open in previous versions, but the annotations do not

display because different objects are used to contain them in Version 7 than before. That is, the annotation objects have never been backward compatible.

Creating Graphical User Interfaces (GUIs)

New Editors for Creating Custom Toolbars within GUIDE

In previous releases, adding toolbars to a GUI had to be done programmatically, by writing code for the `uitoolbar`, `uipushtool` and `uitoggletool` functions. GUIDE now has a Toolbar Editor and an associated Icon Editor that allow you to lay out a toolbar for a GUI and populate it with standard predefined tools for printing, saving, panning, zooming, annotating, etc. or with custom tools that you design yourself. You can draw or modify an icon for any tool on your toolbar with the Icon Editor or import one from an image file.

The Toolbar Editor

When you run GUIDE to create a new GUI or edit an existing one, click the Toolbar Editor icon in the Layout Editor Toolbar or choose **Toolbar Editor** from the **Tools** menu to open the Toolbar Editor. To add tools to the toolbar of your GUI, you drag standard or custom tool icons to the toolbar layout area at the top. You can modify a tool's properties using the Toolbar Editor and Property Inspector. You can also create and customize icons for tools using the new Icon Editor, described next. You control the behavior of a tool in your toolbar with its `ClickedCallback` (and for toggle tools, their `OnCallback` and `OffCallback`) in the GUI's associated M-file. If you use an existing tool, such as Print or Save, you do not need to modify its callback; it is predefined as `%default`.

The Icon Editor

The Icon Editor lets you customize the icons for existing tools or create entirely new icons. Icons are stored as `CData` for each tool, and can also be read from and saved to `.icn` files. The Icon Editor also includes a Color Editor, a palette you can use for picking predefined colors and defining new ones.

Coordinate Readouts in Layout Editor

The GUIDE Layout Editor now has two fields in its lower left corner that continuously provide numeric feedback for:

- **Current Point** — The cursor location within the layout window
- **Position** — The Position property of the currently selected object(s) in the layout window

Both measurements are given in pixels, regardless of the current `Units` settings for GUI objects. When multiple objects are selected, any elements of their `Position` properties that are not identical are read out as `MULTI`. The readouts help you to precisely size and align GUI components.

Documentation for Making GUIDE GUIs Interact

A new documentation section, Making Multiple GUIs Work Together, has been added to illustrate how multiple GUIDE GUIs can work together by sharing data to create a more complicated GUI. It first summarizes the techniques that GUIDE GUIs can use to share data with one another. It then steps through two examples, accompanied by their M-files and FIG-files, to show how to use those techniques for specific tasks.

Functions Being Removed

Function Name	What Happens When You Run the Function?	Use This Function Instead	Compatibility Considerations
`axlimdlg`	Errors	None	No replacement
`cbedit`	Errors	`guide`	Use the `guide` command instead
`clruprop`	Errors	`rmappdata`	Replace existing instances of `clruprop` with `rmappdata`
`ctlpanel`	Errors	`guide`	Use the `guide` command instead
`edtext`	Errors	Set the `editing` property on the text object	No replacement
`extent`	Errors	Get the `extent` property of the text object	No replacement
`getuprop`	Errors	`getappdata`	Replace existing instances of `getuprop` with `getappdata`
`hthelp`	Errors	`web`	Replace existing instances of `hthelp` with `web`
`layout`	Errors	None	No replacement
`matq2ws`	Errors	None	No replacement
`matqdlg`	Errors	None	No replacement
`matqparse`	Errors	None	No replacement
`matqueue`	Errors	None	No replacement
`menubar`	Errors	Set the `menubar` property of the figure to `none`	No differences
`menuedit`	Errors	`guide`	Use the `guide` command instead
`pagedlg`	Errors	`pagesetupdlg`	Replace existing instances of `pagedlg` with `pagesetupdlg`
`setuprop`	Errors	`setappdata`	Replace existing instances of `setuprop` with `setappdata`
`umtoggle`	Errors	Set the `Checked` property of uimenu objects	No differences
`wizard`	Errors	`guide`	Use the `guide` command instead
`ws2matq`	Errors	None	No replacement

External Interfaces/API

Support for 64-bit mxArrays

MATLAB Version 7.5 (R2007b) supports 64-bit mxArrays. This change allows C/C++ and Fortran files built on 64-bit platforms to handle large data arrays.

In earlier versions of MATLAB, `mxArray`s are limited to 2^{31}-1 elements. In Version 7.5 (R2007b) your mxArray can have up to 2^{48}-1 elements.

The `mex` command option, `-largeArrayDims`, uses the large-array-handling mxArray API. Use `mwSize` to represents size values, such as array dimensions and number of elements. Use `mwIndex` to represent index values, such as indices into arrays.

Compatibility Considerations

MEX-files that built properly in previous versions of MATLAB continue to build in Version 7.5 (R2007b).

The default option for `mex` is `-compatibleArrayDims`. If you use this option, `mex` builds the files using the 32-bit array-handling API.

To work with 64-bit mxArrays, your C, C++ and Fortran source code must comply with the 64-bit array-handling API. To use the API to create C/C++ MEX-files, see Handling Large mxArrays . To use the API to create Fortran MEX-files, see Handling Large mxArrays .

Fortran MEX-Files Will Require mwSize and mwIndex

In a future version of MATLAB, the default `mex` command option will change to `-largeArrayDims`. Fortran MEX-files will be required to use the `mwSize` and `mwIndex` preprocessor macros

Compatibility Considerations

To make your Fortran MEX-file compatible with the `-largeArrayDims` option, and to create platform-independent code, you need to include the `fintrf.h` header file in your Fortran source files, and you need to name your source files with an uppercase `.F` file extension. For information on creating MEX-files, see the Gateway Routine in the Fortran MEX-Files topic.

Changes to the MATLAB Locale Setting

Retrieving and using the proper locale setting is a mandatory operation in creating and using applications for international audiences. In R2007b, MATLAB Version 7.5 standardizes the way it initializes the locale setting across platforms. As a result, on Microsoft Windows platforms, MEX-files that use C/C++ locale-dependent standard library functions should note that MATLAB now sets the locale using the `setlocale` function.

Because of changes to Microsoft Visual Studio, MEX-Files created on Windows with dMicrosoft Visual Studio 2003 will not have the same locale setting as MATLAB Version 7.5 (R2007b).

Note C/C++ users must not change the locale setting using the `setlocale` function. This is true for all versions of MATLAB on all platforms.

Changes to MEX Error-Handling Functions mexErrMsgTxt and mexErrMsgIdAndTxt

In MATLAB Version 7.5 (R2007b). the `mexErrMsgTxt` and `mexErrMsgIdAndTxt` functions determine where the error occurred, and display the function name. In previous versions, these functions only display the error message.

For example, if an error occurs in the function `foo`, `mexErrMsgTxt` and `mexErrMsgIdAndTxt` display the following information before the error message:

```
??? Error using ==> foo
```

Rebuild MEX-Files Created with MATLAB Versions Earlier Than V7 (R14)

To work with MATLAB V7.5 (R2007b), MEX-files compiled on any platform with MATLAB versions earlier than V7 (R14) no longer load correctly and must be rebuilt.

Changes to Compiler Support

The set of compilers that MATLAB supports has changed in MATLAB Version 7.5 (R2007b). For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

- “New Compiler Support” on page 30-34
- “Discontinued Compiler Support” on page 30-34
- “Compiler Support to Be Phased Out” on page 30-35

New Compiler Support

MATLAB V7.5 (R2007b) supports new compilers for building MEX-files.

Windows platforms

- Microsoft Visual Studio 2005 SP1

Sun Solaris SPARC (64-bit) platform

- gcc / g++ Version 4.1.2

Discontinued Compiler Support

The following compilers are no longer supported.

Windows platforms

- Microsoft Visual Studio 2005 without SP1

Solaris SPARC (64-bit) platform

- gcc / g++ Version 3.2.3

Compatibility Considerations

To ensure continued support for building your C/C++ programs, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Compiler Support to Be Phased Out

The following compilers are supported in Version 7.5 (R2007b), but will not be supported in a future version of MATLAB.

Windows (32-bit) platform

- Intel Visual Fortran Version 9.0
- Compaq Visual Fortran Version 6.6
- Intel C++ Version 7.1
- Borland C++Builder 6 Version 5.6
- Borland C++Builder 5 Version 5.5
- Borland C++ Compiler Version 5.5
- Compaq Visual Fortran Version 6.1

Changes to Applications Built with Borland 5.5 or 5.6 C Compilers

MATLAB applications built with Borland Version 5.5 or 5.6 C compilers have changed in MATLAB Version 7.5 (R2007b). MATLAB applications that run under Windows are now implemented as console applications, not Windows applications.

Compatibility Considerations

If you have customized the build process based on one of the `bcc*engmatopts.bat` options files, you must edit this file making changes to the appropriate `LINKFLAGS` statement, as described in the `.bat` file comments.

Environment Variable Required for mex with Microsoft Platform SDK Compiler

When you build a MEX-file, an engine application, or a MAT application on a Windows 64-bit platform using the Microsoft Platform SDK compiler, MATLAB requires that you define the environment variable `MSSdk`. The value of this environment variable is the path to the installation directory for Microsoft Platform SDK. The Microsoft Platform SDK installation program does not commonly define this environment variable. For example, you might set the environment variable as follows:

```
C:\Program Files\Microsoft Platform SDK
```

Environment Variables Required for mex with Intel Visual Fortran 9.0

When you build a MEX-file, an engine application, or a MAT application using Intel Visual Fortran 9.0, MATLAB requires that you define an environment variable for the Windows platform you are using.

Windows (32-bit) platform

Define the environment variable `VS71COMNTOOLS`. The value of this environment variable is the path to the `Common7\Tools` directory of the Visual Studio .NET 2002 or 2003 installation directory. (Intel Visual Fortran requires Visual Studio .NET 2002 or 2003 on 32-bit Windows platforms.) The Visual Studio .NET 2003 installation program commonly defines this environment variable. For example, you might set the environment variable as follows:

```
C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools
```

Windows x64 platform

Define the environment variable `MSSdk`. The value of this environment variable is the path to the installation directory for Microsoft Platform SDK for Windows Server 2003. (Intel Visual Fortran requires Microsoft Platform SDK for Windows Server 2003 on Windows x64 platforms.) The Microsoft Platform SDK installation program does not commonly define this environment variable. For example, the environment variable might have the value

```
C:\Program Files\Microsoft Platform SDK
```

Changes to Handling ActiveX Methods

In MATLAB Version 7.4 (R2007a), a change was made to Microsoft ActiveX methods which was not documented in the release notes. This change is described in “Changes to Handling Microsoft ActiveX Methods” on page 31-35 in the MATLAB Release Notes, Version 7.4 (R2007a).

Changes to Dynamic Data Exchange (DDE) Documentation

In MATLAB V5.1, all development work for the Dynamic Data Exchange (DDE) server and client was stopped. This functionality is no longer documented in V7.5 (R2007b). MathWorks provides, instead, a MATLAB interface to COM technology that is documented in Using COM Objects from MATLAB .

Obsolete Functionality No Longer Documented

Documentation for Dynamic Data Exchange (DDE) is no longer included in External Interfaces.

Compatibility Considerations

If you must support this obsolete functionality, we suggest you print a copy of the External Interfaces chapter “COM and DDE Support (Windows Only)” from MATLAB V7.4 (R2007a) or earlier.

Documentation for Obsolete Functions To Be Phased Out

Documentation for the following functions is included in the MATLAB Function Reference documentation for MATLAB V7.5 (R2007b), but will not be included in a future version of MATLAB.

Obsolete Functions
`ddeadv`
`ddeexec`
`ddeinit`
`ddepoke`

Obsolete Functions
`ddereq`
`ddeterm`
`ddeunadv`

The following syntax for `enableservice`, included in the MATLAB Function Reference documentation for MATLAB V7.5 (R2007b), will be removed from the documentation.

```
enableservice('DDEServer',enable)
```

Compatibility Considerations

If you must support this obsolete functionality, we suggest you print and keep a copy of the relevant MATLAB function reference pages from V7.5 (R2007b) or earlier.

R2007a

Desktop Tools and Development Environment

Startup and Shutdown

New features and changes introduced in Version 7.4 (R2007a) are

Double-Clicking Associated File Type in Explorer Now Opens File in Existing Session of MATLAB Software

When you open a file from Microsoft Windows Explorer whose type is associated with the MATLAB application, the file opens in the appropriate tool in the existing session of MATLAB, if MATLAB is already running, or if not, starts MATLAB. This assumes you associated the file types with MATLAB by accepting the defaults while installing MATLAB, or by changing the associations. For example, in Explorer, double-click an M-file to open the file in the MATLAB Editor/Debugger, or double-click a MAT-file to open the Import Wizard and load the data into the workspace in MATLAB. For details, including how to change file associations, see Associating Files with MATLAB on Windows Platforms.

Compatibility Considerations

In previous versions, when you double-clicked a file in Explorer whose type was associated with MATLAB, the file opened in a new session of MATLAB. The change made in MATLAB Version 7.4 (R2007a) to open the file in an existing session was based on many user requests.

In previous versions, double-clicking a MAT-file associated with MATLAB in Explorer opened a new session ofMATLAB and loaded the data into the workspace. When clicked in the Apple Macintosh Finder, the data loaded into the existing session of MATLAB. Now on Microsoft Windows and Macintosh platforms, the Import Wizard opens in the existing session of MATLAB, and you use it to load the data into the workspace.

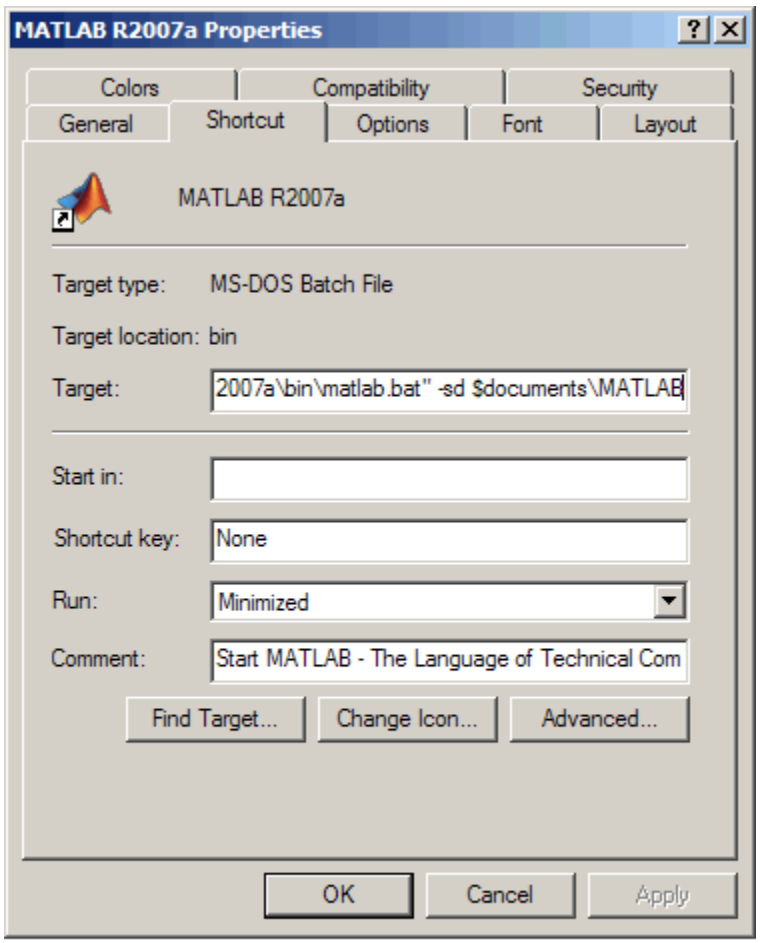
By default, in previous versions, double-clicking an M-file in Explorer opened it in the MATLAB stand-alone Editor. For more information about that change, see “Stand-Alone Editor No Longer Opens By Default; To Be Removed in a Future Version” on page 31-14.

Changes to Startup Directory (Folder) and Startup Options for MATLAB Application on Windows

The default startup directory for MATLAB on Microsoft Windows platforms is now `My Documents\MATLAB`, or `Documents\MATLAB` on the Microsoft Windows Vista platform. Upon startup, MATLAB automatically locates this `MATLAB` folder (creating it if it does not exist), and adds it to the top of the search path. This utilizes standard folders on Windows and Windows Vista platforms to provide a unique startup directory for each user. As such, the folder is part of theWindows (or Windows Vista) user's profile, and will be accessible when the user runs MATLAB from other machines, when the profile is set up to roam.

In the Properties dialog box of the shortcut icon for MATLAB, the **Target** field includes a new startup option `-sd` (for *s*tartup *d*irectory), followed by `$documents\MATLAB`, where MATLAB interprets `$documents` as the `My Documents` folder for the current user (or `Documents` for the Windows Vista platform). If the path for `$documents` is specified in the configuration of the Windows environment via UNC pathname, MATLAB automatically assigns it a mapped drive, creating one if necessary. The `-sd` startup option overrides anything in the **Start in** field, which is now empty by default.

For more information, see Changing the Startup Directory.



Compatibility Considerations

In previous releases, the default startup directory was `\Work`, located in the directory in which MATLAB was installed. For example, in R2006b, if you installed MATLAB in `C:\Program Files`, the default startup directory was `C:\Program Files\MATLAB\R2006b\Work`.

Windows Vista User Account Control (UAC) security features restrict access to `Program Files`. To accommodate this enhanced security model, the default startup directory in MATLAB has been moved outside of `Program Files`.

These are the differences between the startup directory for R2007a and previous releases:

- In previous releases, upon installation, the default startup directory was specified in the **Start in** field of the Properties dialog box for the shortcut icon for MATLAB. You could change the startup directory by replacing the pathname in that field. In R2007a, if you want to specify the startup directory using the **Start in** field, you must also remove from the **Target** field the `-sd` startup option and the pathname that follows it.
- The default startup directory is no longer specific to a release. When you upgrade from R2007a to a future release, files you created and saved in `My Documents\MATLAB` (or `Documents\MATLAB` on Vista) will automatically be in the startup directory for the new release. Previously, you had to move files you created and saved in the `Work` folder for a release, for example, `R2006a\Work`, to the default startup directory for the new release, for example, `R2006b\Work`.
- In previous releases, the default startup directory, `Work`, was shared by all users running that installation of MATLAB. The default startup directory in R2007a, `My Documents\MATLAB` (or `Documents\MATLAB` on Vista), provides each user with a unique startup directory.
- In previous releases, MATLAB added the default startup directory, `Work`, to the bottom of the search path. In R2007a, MATLAB adds the default startup directory, `My Documents\MATLAB` (or `Documents\MATLAB` on the Windows Vista platform), to the top of the search path. In R2007a, for consistency with previous releases, MATLAB adds `...\MATLAB\R2007a\Work` to the bottom of the search path. However, we encourage you to stop using the `Work` folder because support for it might be removed in a future release.

JVM for Windows Updated

MATLAB is now using Sun Microsystems Java (JVM) version 1.5.0_07 on Windows platforms. Java is supplied with MATLAB for Windows platforms, so this change requires no action on your part.

Compatibility Considerations

If you use a specific version of Java with MATLAB on Windows platforms, this change might impact you.

New Version of Java Access Bridge Software

MATLAB now installs Java Access Bridge 2.0, which is used by Freedom Scientific BLV Group JAWS® software for accessibility support.

Confirm Exit Preference to be Enabled by Default in Future Version

When you exit from MATLAB, MATLAB terminates without displaying a confirmation dialog box that asks if you are sure you want to quit. This is the default behavior but you can set a preference to display a confirmation dialog box. In a future version, the confirmation dialog box will appear by default when you quit MATLAB.

Compatibility Considerations

When the confirmation dialog appears by default, you will be able to disable it using preferences. If you have files that include exit statements, you might need to ensure the preference for the confirmation dialog box is disabled.

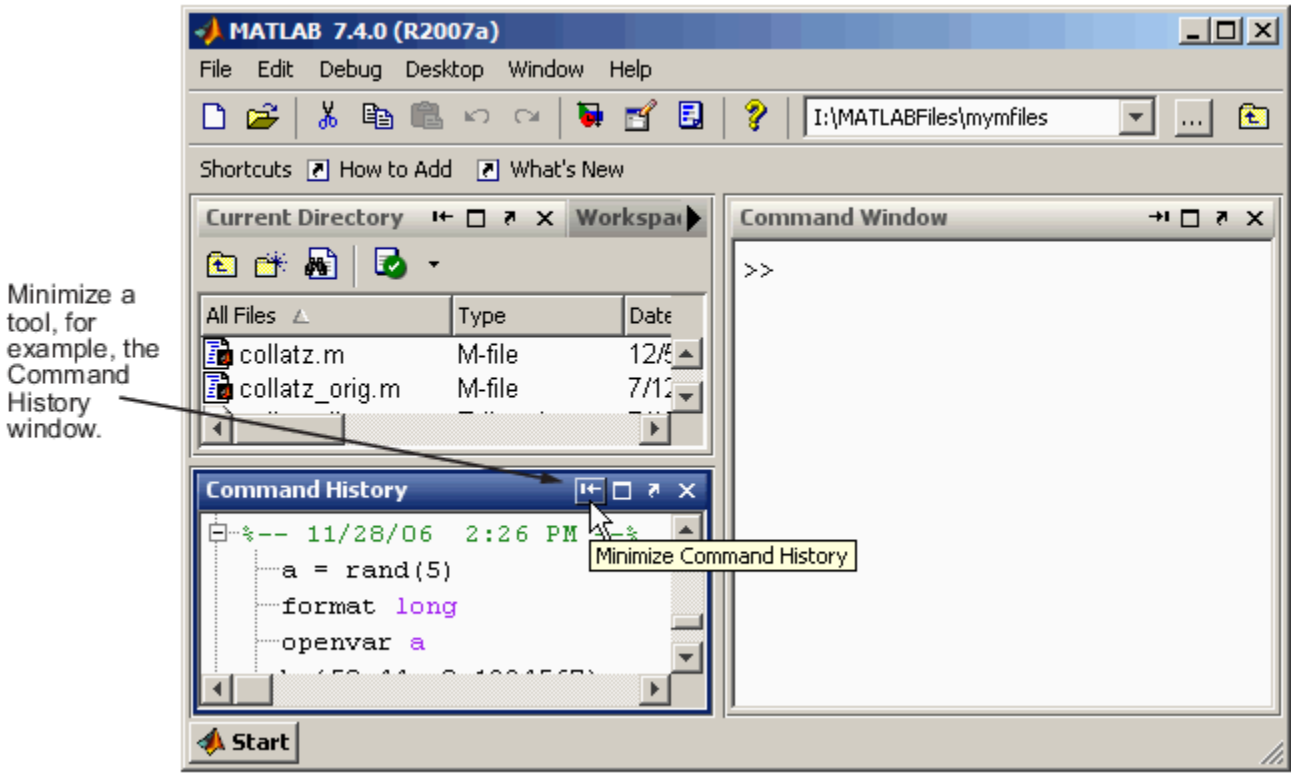
Desktop

New features and changes introduced in Version 7.4 (R2007a) are

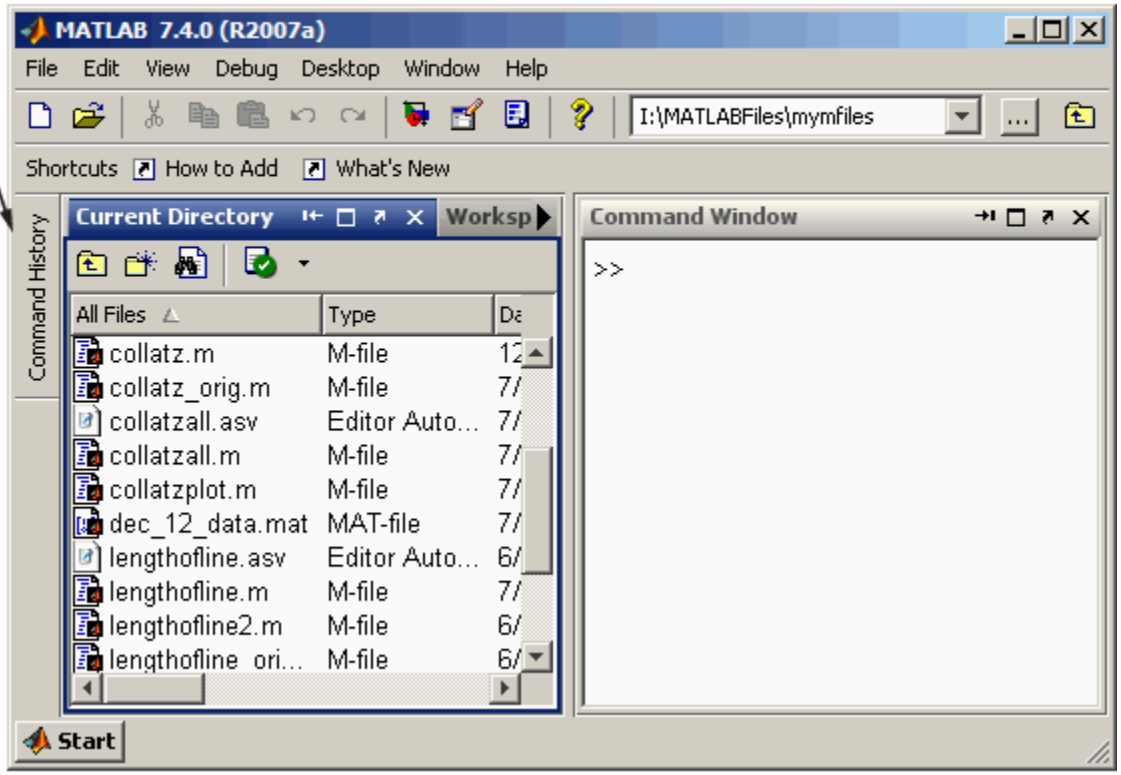
Minimize, Temporarily Display, and Restore Tools in the Desktop

On Windows and UNIX platforms, you can minimize any tool in the desktop. Minimizing a tool creates a button for it along the specified edge. When you want to view the tool, hover over or click the button to temporarily show it in the desktop. Right-click the button and select **Restore *`toolname`*** to return the tool to the desktop. Perform these tasks for the selected tool using items in the **Desktop** menu, equivalent mnemonics (for example **Alt+D, N** to minimize), or buttons on the tool's titlebar.

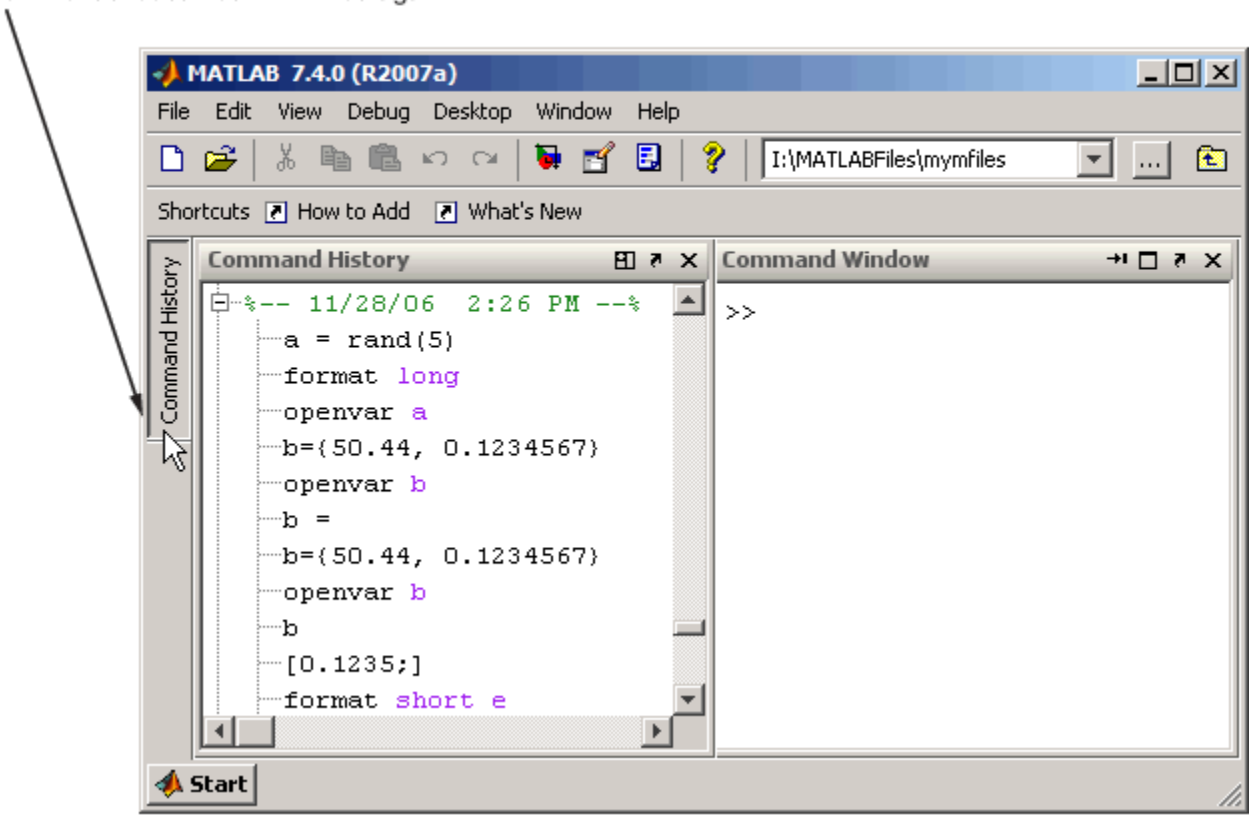
The following illustrations show how to use these features, using the example of the Command History window in the default desktop layout.

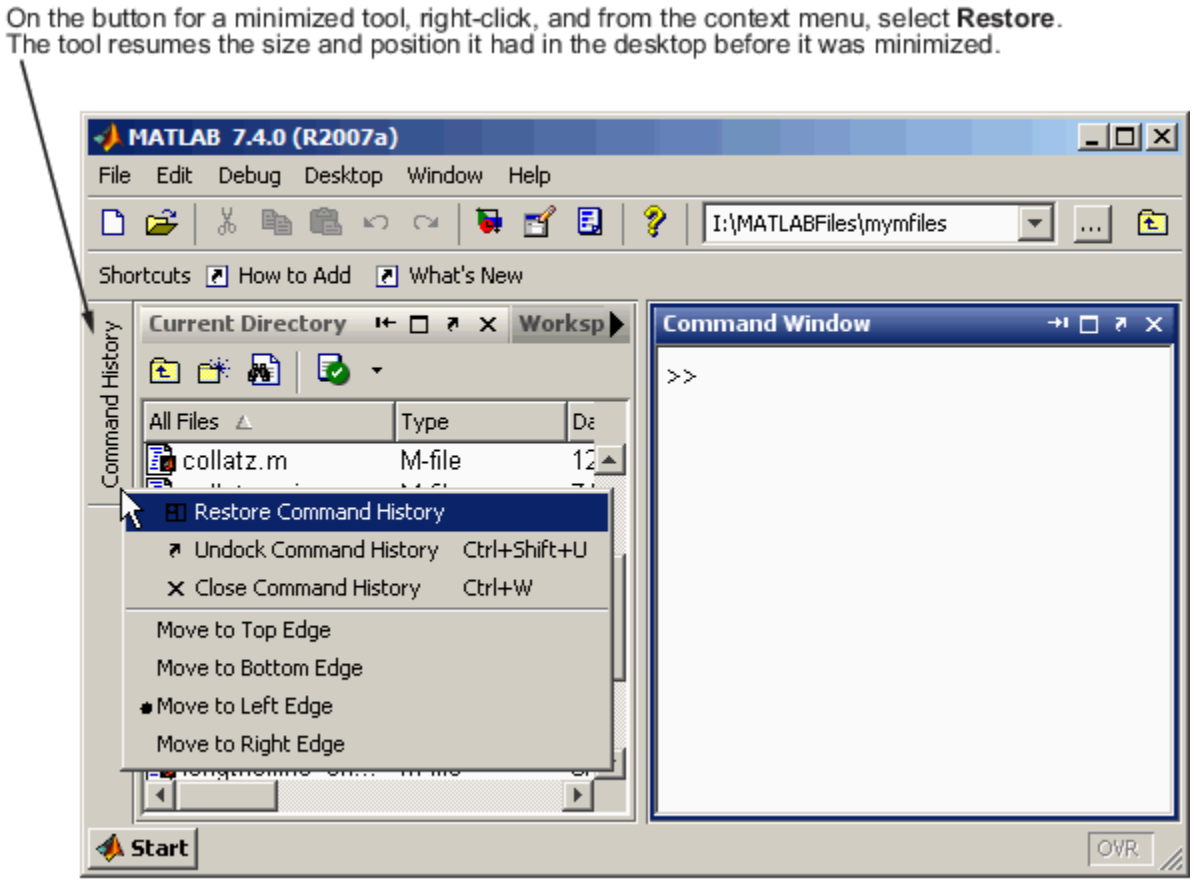


When minimized, a tool, such as the Command Window in this example, is represented by a button on the desktop border.



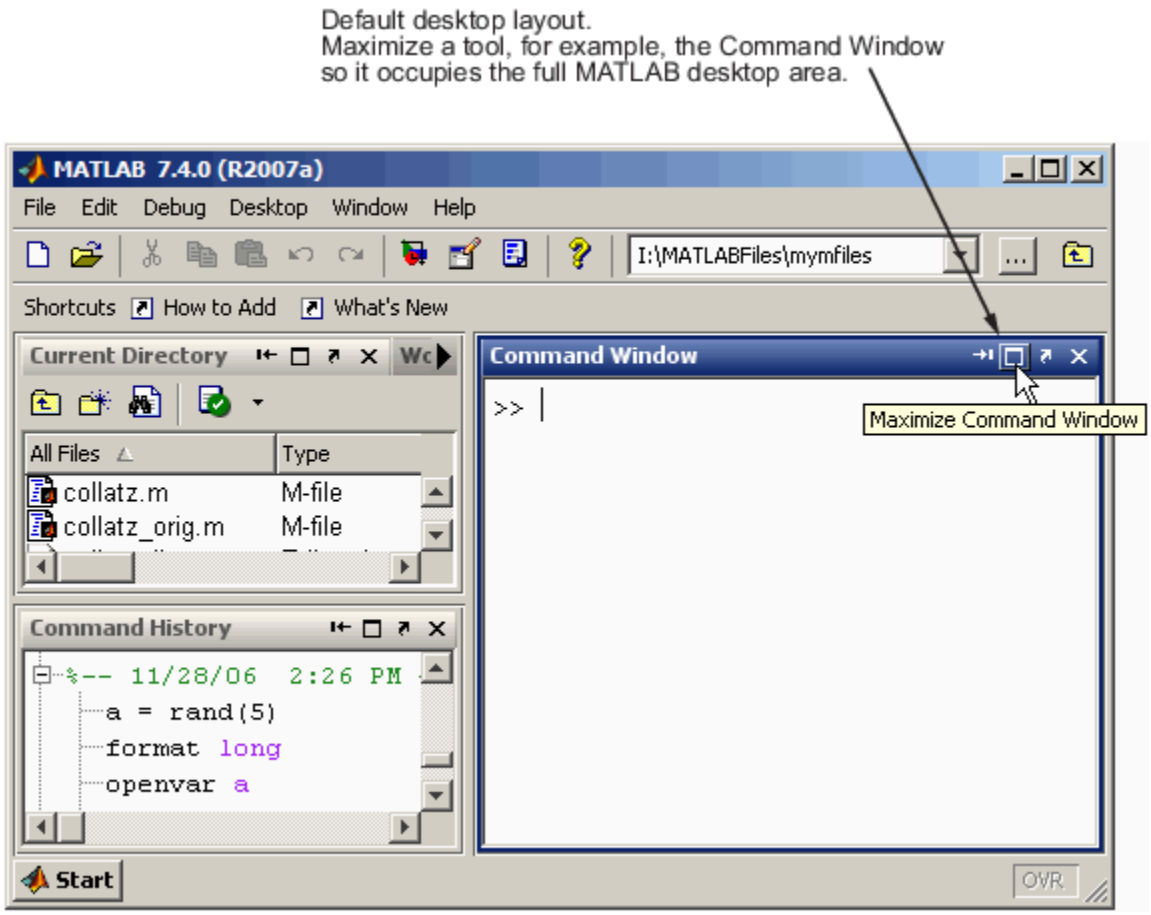
Hover over or click the button for a minimized tool to temporarily view the tool.
The tool is temporarily displayed until you select another tool.
Then the tool becomes minimized again.

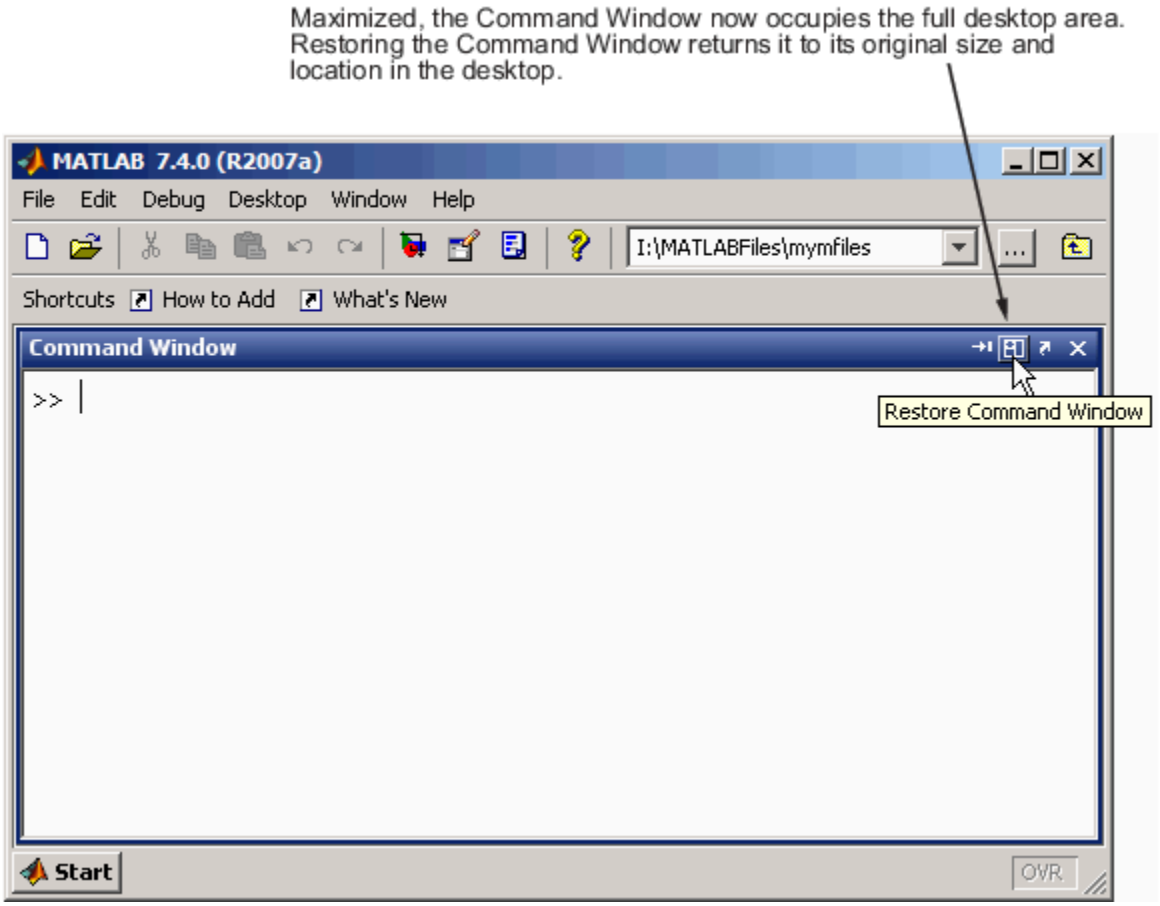




Maximize Tools in the Desktop

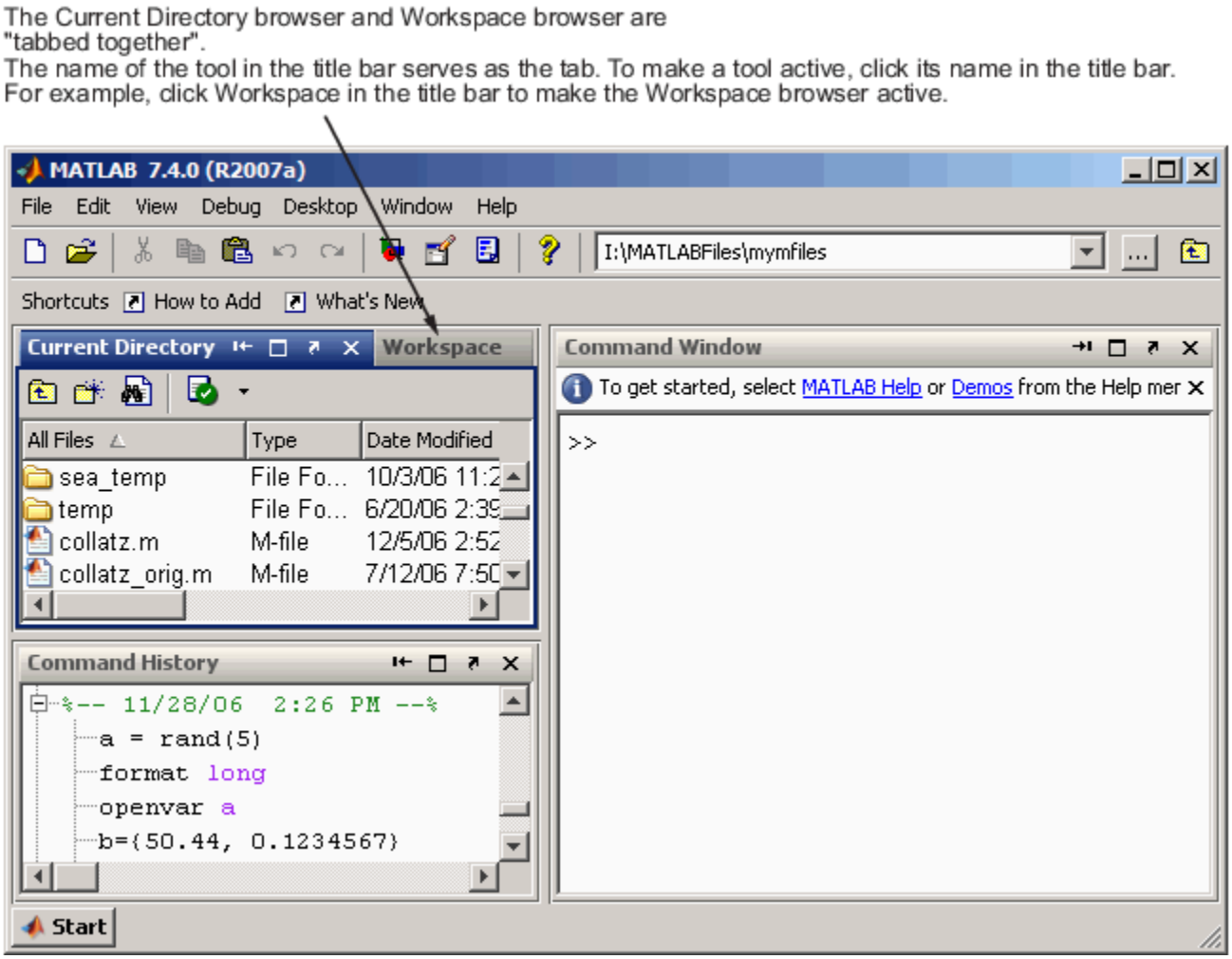
On Windows and UNIX platforms, you can maximize a tool so it occupies the entire desktop tool area in MATLAB, then restore it to return it to its previous location. Perform these tasks for the selected tool using items in the **Desktop** menu, equivalent mnemonics (for example **Alt+D, X** to maximize), or buttons on the tool's titlebar.


Default desktop layout.
Maximize a tool, for example, the Command Window
so it occupies the full MATLAB desktop area.
MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
I:\MATLABFiles\mymfiles
Shortcuts How to Add What's New
Current Directory
All Files
Type
collatz.m M-file
collatz_orig.m M-file
Command History
%-- 11/28/06 2:26 PM
a = rand(5)
format long
openvar a
Command Window
>>
Maximize Command Window
Start
OVR



Tabs for Tools Replaced by Title Bars

In the desktop, when you tab tools together, that is, arrange them so they occupy the same position, the tools' title bars share the title bar area. To make a tool active, select its name in the title bar.



Compatibility Considerations

In previous versions, tools tabbed together each had a tab at the bottom of the area they occupied. The tabs have been removed in favor of tools sharing the titlebar area.

Multithreaded Computation Support Added; Enable Via New Preference

If you run MATLAB on a multiple-CPU system (multiprocessor or multicore), use a new preference to enable multithreaded computation. This can increase performance in MATLAB for element-wise and BLAS library computations.

By default the preference is not set, so you must set it to enable multithreaded computation. With the preference enabled, MATLAB automatically specifies the recommended number of computational threads, although you can change that value. On AMD platforms running the Linux operating system, MATLAB supports multithreaded computation, but requires an extra step to change the default BLAS.

If you are using a multiple-CPU system, you can run the Multithreaded Computation demo to see the performance impact. For more information, see “Enabling Multithreaded Computation” in the MATLAB Desktop documentation.

Running Functions—Command Window and History

New features and changes introduced in Version 7.4 (R2007a) are

Startup Message Bar Replaces Startup Message in Command Window

When you start MATLAB, a getting started message bar appears at the top of the Command Window, that provides links to information that is helpful for new users. You can dismiss the message bar by using the close box in it. Use Command Window preferences to specify whether or not the message bar should appear.

Command History Searching Enhanced

You can now find entries in the Command History window by typing the first few letters of an entry; the previous entry that begins with those letters is selected. Then use up and down arrow keys to extend the find to the next and previous instances. Use the **Ctrl** key with an arrow key to select the current as well as next or previous instances. Use **Ctrl+A** to find and highlight all instances at once. The search does not look at entries in sessions that are collapsed. For more information about this search feature, see Quick Search for Entries Beginning with Specified Letters or Numbers.

Macintosh Platforms—Some Key Bindings in Command Window Changed

On Macintosh platforms, some key bindings were changed to make them more consistent with Mac OS XMac OS X standard behaviors.

Compatibility Considerations

Key bindings you have been used to in the Command Window might have changed.

Help

New features and changes introduced in Version 7.4 (R2007a) are

Demos Now Included in Search Results and Product Filtering

When you perform a search in the Help browser, the results now include code and text found in **Demos**. Also, the **Demos** listing in the Help Navigator pane now synchronizes with the demo currently open when you have the synchronization preference for the Help browser selected. For more information, see Search Syntax and Tips.

When you select **File > Preferences > Help** and enable the Product Filter, only demos for selected products are shown in the **Demos** pane, and searched via the **Search for** field.

Get URL of Displayed Page for Viewing on Web

When viewing a page in the Help browser, select **View > Page Locations**. A window appears providing the location of the current Help page you are viewing. The window provides the page location on both your local system and the MathWorks Web site. Use this feature to

- Send the URL to someone else who wants to view that information and might not have MATLAB or the same version of MATLAB, for example, a colleague or technical support.
- More easily see if this same documentation page has been updated for the latest product version.

Include Search Database for Your Own Help Files

If you create your own HTML help files for use with the MATLAB Help browser, the Help browser can now search the content of your files. Create a search database for your help files using the new `builddocsearchdb` function. The function creates a `helpsearch` directory that contains the search database files. For information on this process, see Making Your HTML Help Files Searchable.

Compatibility Considerations

In previous versions of MATLAB, some users created help search databases for their own help files via assistance from MathWorks Technical Support. If you created a help search database for use with R2006b, it might work in R2007a, but it is recommended that you recreate it for R2007a, even if no content has changed. If you created a help search database prior to R2006b, you must recreate it for R2007a.

Help search databases in prior releases were built from a help `jar` file rather than `html` files. The `builddocsearchdb` function only supports `html` files. However, it is expected to support `jar` files in a future release. If you used a `jar` file for a prior version and want to use a `jar` file in R2007a, build your R2007a help search database using the same technique you used in R2006b.

Video Tutorials Now Accessed Via Web Site

Previously video tutorials were installed when you installed MATLAB. Now the tutorials are on the MathWorks Web site.

Compatibility Considerations

You can still link to and play video tutorials from within the Help browser Demos listings or other links to them in the product and documentation, however this now requires an active Internet connection.

Workspace, Search Path, and File Operations

New features and changes introduced in Version 7.4 (R2007a) are described here.

Current Directory Browser Enhancements

- When you double-click a Windows shortcut in the Current Directory browser, it runs the shortcut.
- When you double-click a `prj` file in the Current Directory browser, it opens in the Deployment Tool.
- You can now find entries in the Current Directory browser by typing the first few letters of an entry; the entry that begins with those letters is selected.

Workspace Browser

When you double-click an object, it opens in the Property Inspector.

Array Editor Enhancements

You can now undo and redo the last operation you performed in the Array Editor. This applies to cut, paste, insert, delete, and clear contents features.

Search Path Changes

On Windows platforms, MATLAB now adds the default startup directory, `My Documents\MATLAB` (or `Documents\MATLAB` on Windows Vista), to the top of the search path upon startup.

Compatibility Considerations

In previous releases, MATLAB added the default startup directory, `Work`, to the bottom of the search path on Windows platforms. For example, in R2006b, it added `...\MATLAB\R2006b\Work` to the bottom of the search path. In R2007a, for consistency with previous releases, MATLAB adds `...\MATLAB\R2007a\Work` to the bottom of the search path. However, we encourage you to stop using the `Work` folder because support for it might be removed in a future release. For more information, see “Changes to Startup Directory (Folder) and Startup Options for MATLAB Application on Windows” on page 31-2.

Editing and Debugging M-Files

New features and changes introduced in Version 7.4 (R2007a) are

Stand-Alone Editor No Longer Opens By Default; To Be Removed in a Future Version

Starting in this release, by default, double-clicking an M-file in Windows Explorer opens the file in the MATLAB Editor/Debugger rather than in the MATLAB stand-alone Editor. In a future version, the stand-alone Editor will not be provided with MATLAB.

Compatibility Considerations

The change to open M-files in the MATLAB Editor/Debugger rather than the stand-alone Editor was made based on many user requests. You can still use the MATLAB stand-alone Editor in this release by starting the application located at: *`matlabroot`*`\bin\win##\meditor.exe`.

Starting MATLAB by double-clicking an M-file requires a MATLAB license, while starting the stand-alone Editor does not require a MATLAB license.

If you want to associate M-files so that when you double-click them they open in the MATLAB stand-alone Editor rather than in the MATLAB Editor/Debugger, follow the instructions in Associating Files with MATLAB on Windows Platforms; rather than specifying the executable for MATLAB, `matlab.exe`, specify `meditor.exe`, which is in the same folder.

Instead of the stand-alone Editor, you can use the MATLAB Editor/Debugger. It provides all the features of the stand-alone Editor plus some not found in the stand-alone Editor, such as debugging and tab completion; for a full list, see “Stand-Alone Editor Will Not Be Included in Next Version” on page 30-14.

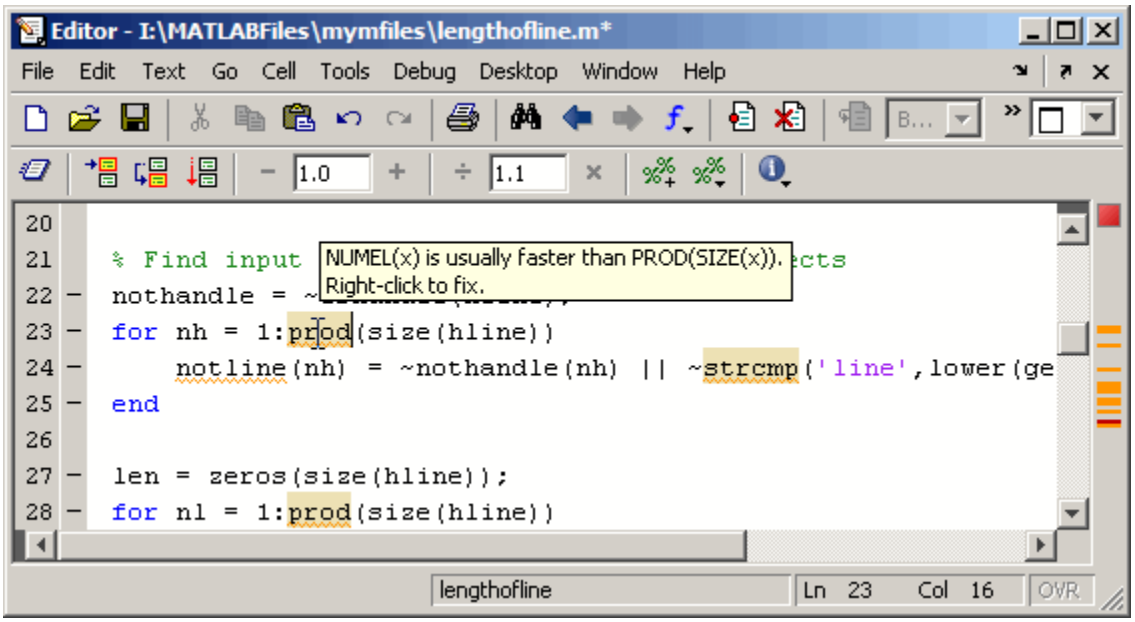
Some users have preferred the stand-alone Editor to the MATLAB Editor/Debugger because of slightly better startup performance and because it does not require a license for MATLAB. For those situations, you can use any text editor you have, such as UltraEdit or Emacs. If you have concerns about the pending removal of the stand-alone Editor, please contact us at `editor-feedback@mathworks.com`, so we can plan to minimize transition issues.

Delimiter Matching Extended to Include Language Keyword Pairs

You can now see the match to language keyword pairs using delimiter matching features that previously existed for parentheses and brackets. For example, when you type `end`, the Editor/Debugger highlights the matching `if`. To set delimiter matching preferences, select **File** > **Preferences** > **Keyboard** > **Delimiter Matching**; click **Help** for more information.

M-Lint Automatic Correction Feature

For some types of warnings or errors, M-Lint can apply an automatic fix to the code. Code that can be automatically corrected appears with a different background color. To perform the fix, right-click the code that is highlighted (for a single single-button mouse, use **Ctrl**+click); from the resulting context menu, select the auto-fix action.



For more information, see step 8 in the example at Automatically Check Code in the Editor — Code Analyzer.

M-Lint Detection of Missing End-of-Line Semicolons Enhanced

In previous versions, there was a single output message generated by a line with a missing terminating semicolon:

```
Terminate statement with semicolon to suppress output.
```

However, M-Lint suppressed displaying the message for files with three or more cells (a cell being indicated by a line with an initial %%). This was done based on the assumption that such files were demo programs, and therefore display of the output was intentional.

In MATLAB Version 7.4 (R2007a), M-Lint no longer makes a distinction for files with three or more cells. Instead, M-Lint displays one message for scripts and a different message for functions:

```
Terminate statement with semicolon to suppress output (in functions).
Terminate statement with semicolon to suppress output (in scripts).
```

The corresponding M-Lint tags to suppress display of such messages within a line are `%#ok<NOPRT>` for functions, and `%#ok<NOPTS>` for scripts.

By default, both of the messages are initially selected in Preferences.

Compatibility Considerations

In MATLAB Version 7.3 (R2006b), `%#ok<NOPRT>` was the only tag used to suppress the missing terminating semicolon. The tag will remain valid for function files, but will not be valid for script files. If you added any `%#ok<NOPRT>` tags in script files in R2006b, change those tags to the new tag `%#ok<NOPTS>`.

Macintosh Platforms—Some Key Bindings in Editor/Debugger Changed

On Macintosh platforms, some key bindings were changed to make them more consistent with standard behaviors for Mac OS X.

Compatibility Considerations

Key bindings you have been used to in the Editor/Debugger might have changed.

Other Changes in the Editor/Debugger

- There is a new confirmation preference for displaying a warning about exiting debug mode in order to save a file.

Tuning and Managing M-Files

There was a minor change in M-Lint behavior—for details, see “M-Lint Detection of Missing End-of-Line Semicolons Enhanced” on page 31-16.

Compatibility Considerations

See the compatibility considerations associated with “M-Lint Detection of Missing End-of-Line Semicolons Enhanced” on page 31-16.

Publishing Results

New features and changes introduced in Version 7.4 (R2007a) are

Publishing Function M-Files Now Supported

You can now publish an M-file function. When publishing an M-file function, you cannot evaluate the code. This feature effectively allows you to save M-file functions to output formats such as HTML or Microsoft Word documents, with formatting. To publish an M-file function, first clear the **Evaluate code** preference in **Editor/Debugger Publishing Preferences**, or run the `publish` function with the `evalCode` option set to `false`.

Specify Maximum Number of Output Lines in Published File

Using the `publish` function, you can specify the maximum number of lines in a published file. Set the new `maxOutputLines` field to a nonnegative value. The default value is `Inf`.

New Publishing Options

New publishing options provide increased flexibility and control over the appearance of the published document. The added options are for adding inline links, inline links with link text, graphics, and HTML markup. See Mark Up MATLAB Comments for Publishing for more information.

Mathematics

New Functions

Function	Description
`bsxfun`	Applies an element-by-element binary operation to two full arrays with singleton expansion enabled
`ilu`	Performs the sparse incomplete LU factorization

More Efficient Matrix Multiplication for Sparse Matrices

Matrix multiplication for `A'*b` now handles sparse matrices more efficiently.

rand Function Uses the Mersenne Twister Algorithm as Default

The `rand` function now uses the Mersenne Twister algorithm as default generator algorithm. This method generates double precision values in the closed interval [2^(-53), 1-2^(-53)], with a period of (2^19937-1)/2. Prior to this release, MATLAB used an algorithm known as the Subtract-with-Borrow (SWB) algorithm.

The `rand` function now produces different results than in previous releases. However, the results returned are still pseudorandom values drawn from a standard uniform distribution. Because the values returned by `rand` are intended to be random, this change should not affect your code.

Compatibility Considerations

There are several things to keep in mind regarding this change:

- If your code requires the exact values returned by `rand` in previous releases, you should use the statement

  ```
  rand('state',0);
  ```

  to reset `rand` to use the SWB algorithm. The new default algorithm's internal state at startup is equivalent to using the statement

  ```
  rand('twister',5489);
  ```

  Note that the `'state'` keyword corresponds specifically to the SWB algorithm; it cannot be used generally to refer to the internal state of the currently active algorithm.
- Existing code that uses the `'state'` keyword to reinitialize `rand` in a statement such as

  ```
  rand('state',sum(100*clock))
  ```

  causes `rand` to switch to using the SWB algorithm. You may want to use the `'twister'` keyword, which resets `rand` but does not switch algorithms.
- Existing code that uses the `'state'` keyword to save and restore the internal state of `rand` in statements such as

```
savedState = rand('state');
 ...                            % code that uses rand
rand('state',savedState);
```

may no longer work as intended. Specifically, the first line of code saves the state of the SWB generator algorithm (see the `rand` documentation for details). If the default Mersenne Twister algorithm was the active one at that time, then using the saved state in the last line does not restore the `rand` internal state to its original conditions. Instead, it switches the `rand` algorithm to SWB. To save and restore the internal state of the Mersenne Twister algorithm, use the `'twister'` keyword.

Upgrade to BLAS Libraries

MATLAB now uses new versions of the Basic Linear Algebra Subroutine (BLAS) libraries. For Windows, Intel Mac, and Intel processors on Linux platforms, MATLAB supports the Intel Math Kernel Library (MKL) version 9.0. For AMD processors on Linux platforms, MATLAB uses the AMD Core Math Library (ACML) version 3.5. For the Solaris platform, MATLAB uses the Sun Performance Library from Sun Studio 11.

mode of Empty Array Now Returns NaN

The `mode` function, when operating on an empty array (`[]`), returns a 1-by-0 array in previous releases, while related functions `mean`, `median`, `std`, and `var` return `NaN` when given the same input. In this release, `mode` returns `NaN` for an empty array input.

Compatibility Considerations

Existing program code that relies on mode of an empty array to return an empty array should be modified.

Change to Syntax for Setting BLAS Library Version on Linux

If you change the BLAS library used by MATLAB on Linux platforms, MATLAB now loads libraries in the left-to-right order specified in the syntax. For example, to load the Intel MKL BLAS, from a system prompt, run

```
setenv BLAS_VERSION mkl.so:mklcompat.so
```

MATLAB loads `mkl.so` first, and then loads `mklcompat.so`.

This also applies if you edit `bin\$(ARCH)\blas.spec` directly.

Compatibility Considerations

This syntax differs from that used for Linux platforms in prior versions.

Data Analysis

Programming

New Functions

New Functions in this release are

Function	Description
`assert`	Generates an error when a specified condition is violated. Displays either the default error message or one that you have written.
`ismac`	Returns `true` if you are currently running one of the Mac OS X versions of MATLAB.
`verLessThan`	Compares the specified version number and toolbox name with the version of that same toolbox that is currently running. Use `verLessThan` in your functions when you want to write code that can run across multiple versions of MATLAB.

Parse Inputs with Consistently Implemented Mechanism

The new `inputParser` class provides a consistent mechanism for parsing and validating input arguments in the M-file functions you write. Using `inputParser` methods in the body of your function, you build a schema that represents each valid input argument to the function. In the schema, you can specify whether arguments are required or optional, and if they are to be passed as single values or as parameter-value pairs. The schema also provides a means of validating each incoming argument. The properties of the `inputParser` class give you additional information about arguments that are passed to the function.

For more information, see Parse Function Inputs in the MATLAB Programming documentation.

textscan Returns Like Values in Same Cell Array

In previous versions of MATLAB, the `textscan` function always returned each output value in a separate cell array, even if those values were of the same data type. In this release, if you call `textscan` with the new `CollectOutput` switch, MATLAB returns all consecutive data that is of the same type in the same cell array. This can save you the extra task of sorting through the output and merging like data types together in your own code.

For more information, see the `textscan` function reference page.

Numbered Arguments for Formatted String Functions

Using numbered argument specification with MATLAB functions that employ format specifiers such as `%d` or `%s` (e.g., `sprintf`, `error`), you can pass the numeric and character string values that correspond to these format specifiers in a varying order. This can be useful when translating format strings in a different sentence structure.

In addition to identifying the values, you can also use numbered arguments to specify the field width and precision of the format specifiers. In the following example, the first format specifier in the `sprintf` command is `%1$*4$f` (quite different from the usual `%f` specifier). The `1$` tells MATLAB that the value that is to replace this format specifier is in the first argument following the format

string; that is, 123.45678. The `*4$` indicates that the field width for this specifier is being passed in the fourth argument following the format string: 15.

In the second and third arguments (`%2$.*5$f` and `%3$*6$.*7$f`), the symbols `2$` and `3$` represent the values, `*5$` and `*7$` represent precisions, and `*6$` represents field width:

```
sprintf('%1$*4$f   %2$.*5$f   %3$*6$.*7$f', ...
123.45678, 16.42837, pi, 15, 3, 6, 4)

ans =
     123.456780   16.428   3.1416
```

For more information, see Formatting Strings in the MATLAB Programming documentation.

The dir Function Returns Additional datenum Field

The `dir` function now returns the date the file or directory was last modified in two formats: string and numeric. The numeric date value is not specific to any particular locale, and thus is compatible for international use:

```
fileinfo = dir('myfile.txt')

fileinfo =
       name: 'myfile.txt'
       date: '16-Mar-2006 13:34:01'
      bytes: 251
      isdir: 0
    datenum: 7.3275e+005
```

This new output is also returned when running `dir` on an FTP server.

Using whos -file on Objects with Overloaded size or class Methods

MATLAB is unable to determine the true size of an object stored in a MAT-file if the class of this object overloads the MATLAB `size` function. Likewise, MATLAB cannot determine the true class name of an object if it overloads the MATLAB `class` function. For these reasons, in previous versions of MATLAB, the command `whos -file` does not return or display object size and class accurately if the class of that object overloads these methods, but instead always returns `1x1` for the size and a default name for the class.

In this release, `whos -file` returns the empty matrix (`[]`) or displays a hyphen (`-`) for objects that overload `size`, and returns and displays `unknown` for objects that overload `class`.

Compatibility Considerations

If you use `whos -file` on objects in any of your programs, and if any of these objects overloads the `size` function, then you need to be aware that MATLAB now returns `[]` instead of a 2–element vector of ones in the `size` field of the output structure.

mat2str Returns Correct Output for Strings

The documentation for the `mat2str` function states that the `str` output of this function “is suitable for input to the `eval` function such that `eval(str)` produces the original matrix to within 15 digits

of precision.” The behavior of `mat2str` when given a character array as input, however, did not abide by this rule. This inconsistency has been fixed in this release.

In MATLAB Version 7.3, the `eval` command below generated an error.

```
s = mat2str('MATLAB')
s =
   'MATLAB'

eval(s)
ans =
   MATLAB
```

Compatibility Considerations

You might have to modify M-file programs that expect the previous behavior from `mat2str`.

Warning Generated by try-catch

To accommodate future changes in the MATLAB error-handling capabilities, there is a new restriction to the syntax of the `try-catch` block. When the first MATLAB statement that follows the `try` keyword consists of just a single term (e.g., `A` as opposed to `A+B`) occurring on the same line as the `try`, then that statement and the `try` keyword should be separated by a comma. For example, the line

```
try A
```

should be written as either

```
try, A
```

or on two lines as

```
try
   A
```

This affects only single-term statements. For example, the following statement continues to be valid:

```
try A+B
```

The same holds true for the `catch` keyword and a single-term statement following the keyword on the same line. A valid `try-catch` statement of this type should be composed as follows:

```
try, A, catch, B, end
```

If you omit the commas following try and/or catch, your code will continue to operate correctly. However, MATLAB will issue a warning:

```
try statements, catch statements, end

Warning: This try-catch syntax will continue to work in R2007a,
but may be illegal or may mean something different in future
releases of MATLAB.
```

As with previous releases, the recommended syntax for a `try-catch` block is as follows:

```
try
   try_statements
catch
   catch_statements
end
```

Note Due to a bug in the R2007a release, the warning for a `catch` followed immediately by a single term is thrown as an error, even though the text of the message says that it is a warning.

Compatibility Considerations

Your M-file programs may generate the warning shown above if the correct syntax for `try` and `catch` is not used.

save -regexp Saves to Correct Filename

In previous releases, the following command mistook the `-regexp` argument (`[ab]` in the case shown below) to be the name of the file to save to:

```
save -regexp [ab]
```

In this release, the filename is correctly understood to be the default `save` filename, `matlab.mat`, and any arguments that follow the `-regexp` flag and precede the next flag are interpreted as patterns to be matched.

Functions Not Callable If in Directory Under Class Directory

M-files placed in a directory that is one or more levels beneath a MATLAB class directory are not callable from that same directory. A MATLAB class directory contains methods of a class and has a filename that begins with the @ character.

In the following example, function `myfun1` is not callable if the current working directory is set to `dir1`. The same holds true for `myfun2` if the current working directory is set to `dir2`.

```
\home\matlab\@myobj\dir1\myfun1.m
\home\matlab\@myobj\dir1\dir2\myfun2.m
```

This behavior existed in the R2006b release, as well as in this release.

Compatibility Considerations

You will no longer be able to call a file that is one or more levels beneath a MATLAB class directory if your current working directory is set to that same directory.

Improved Performance on Certain Platforms and Operations

As of release R2006b, MATLAB offers improved performance in the following areas:

- Improved performance on 64-bit Windows XP and Linux platforms. This is independent of the size of data set in use.

- Faster scalar indexing into cell arrays.
- Faster assignment of cell array data to variables.

Technique to Conserve Memory on Windows Vista

To conserve memory on machines running Windows Vista, you can reduce the amount of virtual memory space reserved by the operating system by using the command:

```
BCDEdit /set increaseuserva 3072
```

More documentation on this option can be found at the following URL: `http://msdn2.microsoft.com/en-us/library/aa906211.aspx`.

ispuma Function Deprecated

Because MATLAB no longer supports Mac OS X 10.1, the `ispuma` function always returns `false` in this release, and also displays a warning message that the function is now deprecated. Support for `ispuma` will be removed in a future release of MATLAB.

Compatibility Considerations

It is recommended that you remove calls to `ispuma` from your M-file functions.

Graphics and 3-D Visualization

Nudging Annotations with Arrow Keys

In plot edit mode, annotations such as textboxes, lines, arrows, doublearrows, and text now respond to pressing directional arrow keys. Each keypress will move the selected annotation(s) one or more pixels in the indicated direction. If you select multiple objects, they all move together in response to arrow key strokes. Normally, selected objects move one pixel with each press of an arrow key. If you have selected **Snap to Layout Grid** from the **Tools** menu, each keypress makes objects move to the next grid position.

Movies No Longer Play During Loading

The `movie` function no longer plays each frame one extra time. Previously, it would show frames as they were loaded into memory to speed up display. This behavior is no longer required and has been eliminated.

Compatibility Considerations

If you have code that calls `movie` in a loop a certain number of times and want that number to remain the same, you need to add one iteration to the loop.

Ghostscript Printing Software Upgraded

The Ghostscript software used by some print devices has been upgraded from an older version to Version 8.54. Starting in this release, MathWorks is no longer shipping a separate, standalone Ghostscript executable program.

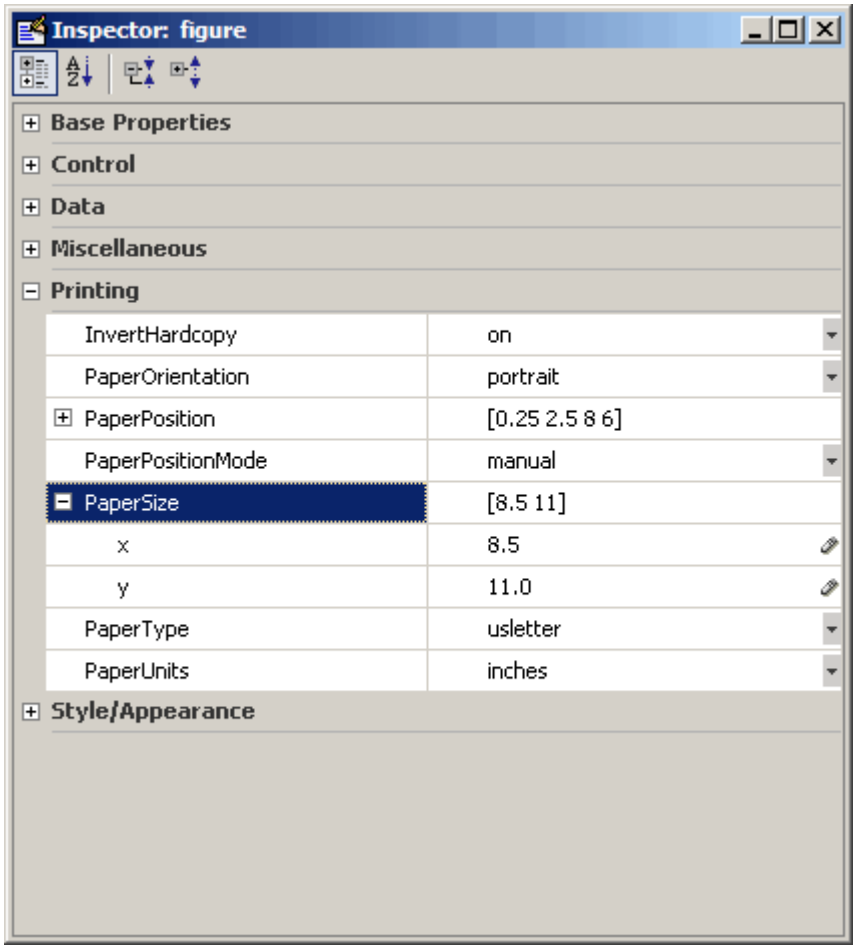
Compatibility Considerations

If you have written M-code based on undocumented functions that called the old version of Ghostscript, your code will no longer work.

As a consequence of this upgrade, support for printing on DEC LN03 printers (`print –dln03`) has been removed from MATLAB.

Property Inspector Now Categorizes Graphic Object Properties

The Property Inspector (accessed via the Property Editor, GUIDE, and the `inspect` function) now provides tree views of groups of graphic object properties as well as an alphabetical list of all properties. You can switch between views via the two buttons at the top of the window, as shown in the following picture of the tree view of the Printing category:



In addition, the type of object currently being inspected is now shown in the title bar of the Property Inspector window. Formerly it was shown in the gray area below the title bar, which now contains the view manipulation buttons.

To change view, click a button without the blue background (the collapse and expand buttons at right are disabled in alphabetic list view). When you change views, the selected property stays the same. To inspect properties of graphic objects in the tree view, click the + icon next to a category of interest. Do the same to open properties within a category that have multiple elements. Note that only Handle Graphics objects have categories; annotation and most other MATLAB objects can be displayed only in alphabetic list view.

The functionality of the Property Inspector has not changed; only its GUI has. As previously, if you select a set or array of objects and inspect them, the Property Inspector only displays those properties that all the objects share.

For details on using the Property Inspector, see Accessing Object Properties with the Property Inspector in the MATLAB Graphics documentation and the inspect reference page.

Compatibility Considerations

Text that you type or delete after clicking a text field that has a text widget icon (for example, `XTickLabel`), will not persist unless you press **Return** before clicking elsewhere. This mimics the

behavior of clicking the text widget icon to open a text entry dialog box, typing into it, and accepting the result by clicking **OK**. This behavior differs from that which existed prior to R2006b (MATLAB V. 7.3).

Figure WindowScrollWheelFcn Property Programs Scrollwheel Events

The new figure property `WindowScrollWheelFcn` enables you to write a callback function that handles mouse wheel scrolling events while the figure has focus. See the documentation for the figure `WindowScrollWheelFcn` for more information.

Figure KeyReleaseFcn Property Programs Key Release Events

The new figure property `KeyReleaseFcn` enables you to write a callback function that handles key release events (analogous to `KeyPressFcn`). See the documentation for the figure `KeyReleaseFcn` for more information.

Creating Graphical User Interfaces (GUIs)

GUIDE No Longer Copies Callbacks When You Duplicate Components

In GUIDE, when you copy a component for which one or more of its callback properties are defined, the callback properties of the newly created component are now set to their default values, the ones you get when adding a component directly from the Component Palette. Previously, their values were copied from the corresponding callback properties of the original component. New callback subfunctions can be generated in the GUI M-file for the new component in the same way as for any component in the GUIDE layout.

Compatibility Considerations

If you have relied on the old behavior, in which the callbacks properties of the new component point to the callbacks of the original component, you will now have to explicitly change the callback properties, using the Property Inspector, to do that.

GUIDE M-File-Defined handles Structure Fields No Longer Become Permanent

The `handles` structure that is provided to every GUIDE generated callback subfunction in the GUI M-file is constructed at runtime every time the GUI runs. When the GUI is fully initialized, the `handles` structure contains only handles to all the components in the GUI and custom data added in any `CreatedFcn` callbacks and/or the `OpeningFcn`. Previously, fields that the M-file had added to the handles structure could sometimes become a permanent part of the structure.

Compatibility Considerations

If your GUI runs well but displays error messages about missing fields in the `handles` structure when starting up, you may have to update the code in the GUI M-file where those fields are accessed. You may need to initialize those `handles` fields properly in a `CreateFcn` callback and/or in the `OpeningFcn`.

UNIX: File Dialog 'Location' Property Is Obsolete

For UNIX platforms, the `Location` property of `uigetfile` and `uiputfile` is obsolete. Previously, for UNIX platforms only, you could use the `Location` property to specify the screen location at which the dialog box would originally be displayed.

Compatibility Considerations

Since the UNIX `uigetfile` and `uiputfile` `Location` property is now obsolete, MATLAB ignores any occurrences in existing code of the `Location` property

```
uigetfile(...,'Location',[X Y])
```

or the older use of `X` and `Y` arguments to specify location

```
uigetfile(...,X,Y)
```

A warning is displayed for either syntax, on all platforms.

Functions Becoming Obsolete

The following functions are either obsolete or grandfathered in MATLAB 7.4 (R2007a): `cshelp`, `figflag`, `getstatus`, `menulabel`, `popupstr`,`setstatus`, `hidegui`, `uigettoolbar`.

Compatibility Considerations

The functions shown in the following table will continue to work but their use may generate a warning message. As soon as possible, replace any occurrences you may have of these functions with the function(s) shown in the following table, if any, or with other suitable code.

Function	Replacement
`cshelp`, `figflag`, `getstatus`, `menulabel`, `popupstr`, `setstatus`	Grandfathered. There are no replacements. No enhancements will be made and only critical bugs will be fixed. No warnings will be displayed. These functions may become obsolete if replacements become available.
`hidegui`	Obsoleted. Now issues warnings. Set the figure `HandleVisibility` property instead.
`uigettoolbar`	Obsoleted. No warnings will be displayed until a replacement is available in a future release.

External Interfaces/API

New File Extensions for MEX-Files

MEX-Files in MATLAB for Apple Macintosh (Intel)

With the introduction of MATLAB for Macintosh (Intel), the MEX-files you build have the extension `.mexmaci`. The `mexext` command in MATLAB returns `mexmaci` for Macintosh (Intel).

Compatibility Considerations

MEX-files built using MATLAB for Macintosh PowerPC®, which have `.mexmac` extensions, cannot be used in MATLAB for Macintosh (Intel).

Note All MEX-files on Macintosh platforms need to be recompiled for R2007a.

MEX-Files in MATLAB for 64-bit Sun Solaris SPARC

With the introduction of MATLAB for 64–bit Solaris SPARC®, you can now build 64-bit MEX-files for the Solaris platform. These MEX-files have the extension `.mexs64`. The `mexext` command in MATLAB returns `mexs64` for Solaris SPARC.

Compatibility Considerations

MEX-files built using MATLAB for Solaris 32, which have `.mexsol` extensions, cannot be used in MATLAB for Solaris SPARC.

Note All MEX-files on Solaris 64 platforms need to be recompiled for R2007a.

MEX-Files Built in MATLAB R11 or Earlier Must Be Rebuilt

In order to work with MATLAB V7.4 (R2007a), MEX-files compiled on Microsoft Windows (32–bit) platforms with MATLAB R11 or earlier will no longer load correctly and must be recompiled.

Compatibility Considerations

Recompile these MEX-files with MATLAB R12 or later.

Changes to Compiler Support

The set of compilers that MATLAB supports has changed in MATLAB Version 7.4 (R2007a). For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

- “New Compiler Support” on page 31-32
- “Fortran Compatibility Considerations” on page 31-33
- “Discontinued Compiler Support” on page 31-33

New Compiler Support

MATLAB V7.4 (R2007a) supports new compilers for building MEX-files.

Windows (64-bit) platform

- Intel C++ version 9.1
- Intel Visual Fortran version 9.1

Windows (32-bit) platform

- Intel C++ version 9.1
- Intel Visual Fortran version 9.1
- Microsoft Visual C++ 2005 version 8.0 Express Edition

Note If you use the `mex -f` *`matlabroot`*`/bin/`*`$ARCH`*`/mexopts/msvc80freeopts.bat` switch to build a MEX-file using Microsoft Visual C++ 2005 Express Edition, the environment variable `MSSdk` must be defined. The value of this environment variable is the path to the installation directory for Microsoft Platform SDK for Windows Server 2003. (Microsoft Visual C++ 2005 Express Edition requires Microsoft Platform SDK for Windows Server 2003.)

Macintosh PowerPC and Macintosh (Intel) platforms

- Apple Xcode 2.4.1 (gcc / g++ version 4.0.1)
- g95 version 0.90

Note All MEX-files on Macintosh platforms need to be recompiled for R2007a.

Linux (64-bit) platform

- gcc / g++ version 4.1.1
- g95 version 0.90

Linux (32-bit) platform

- gcc / g++ version 4.1.1
- g95 version 0.90

Note All Fortran MEX-files compiled on Linux platforms need to be recompiled for R2007a.

Solaris SPARC (64-bit) platform

- cc / CC version 5.8
- gcc / g++ version 3.2.3
- f90 version 8.2

Fortran Compatibility Considerations

In R2007a we have added support for a new Fortran compiler g95 on the Linux and Macintosh platforms. This compiler implements the Fortran 95 language standard. It replaces previously supported Fortran compilers which implemented a previous language standard.

This may cause incompatibilities in your Fortran source code for MEX-files.

Compatibility Considerations

Refer to the IBM XL Fortran V10.1 for Linux Language standards Web site `http://publib.boulder.ibm.com/infocenter/lnxpcomp/v8v101/index.jsp?topic=/com.ibm.xlf101l.doc/xlflr/languagestandards.htm` for information about incompatibilities between language standards.

Discontinued Compiler Support

The following compilers are no longer supported.

Linux (64–bit) platform

- gcc / g++ version 3.4.5
- g77 version 3.4.5

Linux (32–bit) platform

- gcc / g++ version 3.4.5
- g77 version 3.4.5

Macintosh PowerPC platform

- gcc / g++ version 3.3
- Absoft f77 / f90 version 8.2a

Compatibility Considerations

To ensure continued support for building your C/C++ programs, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Compiler Support To Be Phased Out

The following compilers are supported in Version 7.4 (R2007a) but will not be supported in a future version of MATLAB.

Windows (32-bit) platform

- Intel C++ version 7.1
- Intel Visual Fortran 9.0
- Borland C++Builder 6 version 5.6
- Borland C++Builder 5 version 5.5
- Borland C++ Compiler version 5.5

- Compaq Visual Fortran version 6.6
- Compaq Visual Fortran version 6.1

Additional Linker Support for Intel Fortran

MATLAB V7.4 (R2007a) supports new linkers for building MEX-files with Intel Visual Fortran 9.0.

Windows (32-bit) platform

- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio 2005

Windows (64-bit) platform

- Microsoft Visual Studio 2005
- Microsoft Platform SDK for Windows Server 2003 (Build 1289 or later)

New COM Features

Programmatically Connect to Instances of a COM Automation Server

MATLAB can now programmatically connect to an instance of a COM Automation server using the new `actxGetRunningServer` function.

Improve the Custom Interface API

Changes to the `actxserver` function allow you to create a COM Automation server using a custom interface.

New COM Data Type Support

Additional COM data type support has been added. See Handling COM Data in MATLAB Software for a description of supported data types.

Enhanced Support for COM Interface Events

MATLAB users can take full advantage of event interfaces provided by COM Automation servers. This change is implemented in the `events`, `eventlisteners`, `isevent`, `registerevent`, `unregisterevent`, and `unregisterallevents` functions.

For an example of this feature, see Responding to Interface Events from an Automation Server.

Get the Status of a MATLAB Automation Server

Using the `enableservice` function you can learn the current state of a MATLAB Automation server. The function returns a logical value, where logical `1` (`true`) means MATLAB is an Automation server and logical `0` (`false`) means MATLAB is not an Automation server.

For example, if you type

```
enableservice('AutomationServer')
```

and MATLAB displays

```
ans =

     1
```

then MATLAB is currently an Automation server.

Changes to MATLAB Version-Specific ProgID

A programmatic identifier, or ProgID, is used to create a COM component. MATLAB's version-specific ProgID Matlab.Application.N.M now let's you specify both a major and minor version number.

For example, to specify MATLAB version 7.4, use the ProgID `Matlab.Application.7.4`.

Changes to Handling Microsoft ActiveX Methods

Beginning in MATLAB Version 7.4 (R2007a), an ActiveX method with the same name as a class is treated as a constructor and cannot be called in the same way as an ordinary method.

In the following example:

```
myApp = actxserver('Excel.Application');
op = invoke(myApp.Workbooks, 'open', 'MyFile.xls');
Sheets = myApp.ActiveWorkBook.Sheets;
target_sheet = get(Sheets, 'item','Sheet1');
invoke(target_sheet, 'Activate');
Activesheet = myApp.Activesheet;
cellname = 'B2';
Range = Activesheet.cells.Range(cellname,cellname);
```

the term `Range` is both a function on the MATLAB path and a constructor of the class `Range`. MATLAB tries to execute the function `range`, which generates the error:

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

Error in ==> MyScript at 8
Range = Activesheet.cells.Range(cellname,cellname);
```

To get the property value, use the `get` function.

For example:

```
Range = get(Activesheet.cells, 'Range', cellname, cellname);
```

R2006b

Desktop Tools and Development Environment

Startup and Shutdown

New features and changes introduced in Version 7.3 (R2006b) are described here.

Associate Files from MATLAB Program with Windows Operating System Using New Utility

You can run a utility from the Help browser to associate `.m`, `.mat`, `.fig`, `.p`, and `.mdl` files with the MATLAB program in the Microsoft Windows operating system. After running the utility, you will be able to start MATLAB by double-clicking any of those file types in Windows Explorer. You can still use Windows Explorer Folder Options to perform these file associations, but the utility makes the task more convenient. For details, see Associating Files with MATLAB on Windows Platforms.

Redirect Output on UNIX Now Sends Errors to Shell

When you start MATLAB on platforms running The Open Group UNIX operating system using the `-nodesktop` startup option, and you redirect output, for example to a file, MATLAB now sends any errors to the shell, while normal output goes to the redirect target. This change was made so that redirection withMATLAB follows standard behavior for the UNIX operating system.

For example:

```
matlab -nodesktop -"r magic(3), magi(5)" > test.txt
```

starts MATLAB in `nodesktop` mode, runs the statement `magic(3)` and writes the output to `test.txt`. When MATLAB runs `magi(5)`, execution fails and MATLAB displays the error message in the shell.

Compatibility Considerations

In previous versions, MATLAB redirected both output and error messages, so in the above example, MATLAB wrote the output of `magic(3)` as well as the error message from `magi(5)` to `test.txt`.

If you have shell scripts that use > to redirect output, and you rely on errors appearing in the output target, you need to modify the `matlab` startup statements in those scripts.

To achieve the former behavior, that is, to redirect both output and errors to the specified target, use that specific redirect syntax for your shell. For example, in tcsh, use >&, as in

```
matlab -nodesktop -"r magic(3), magi(5)" >& test.txt
```

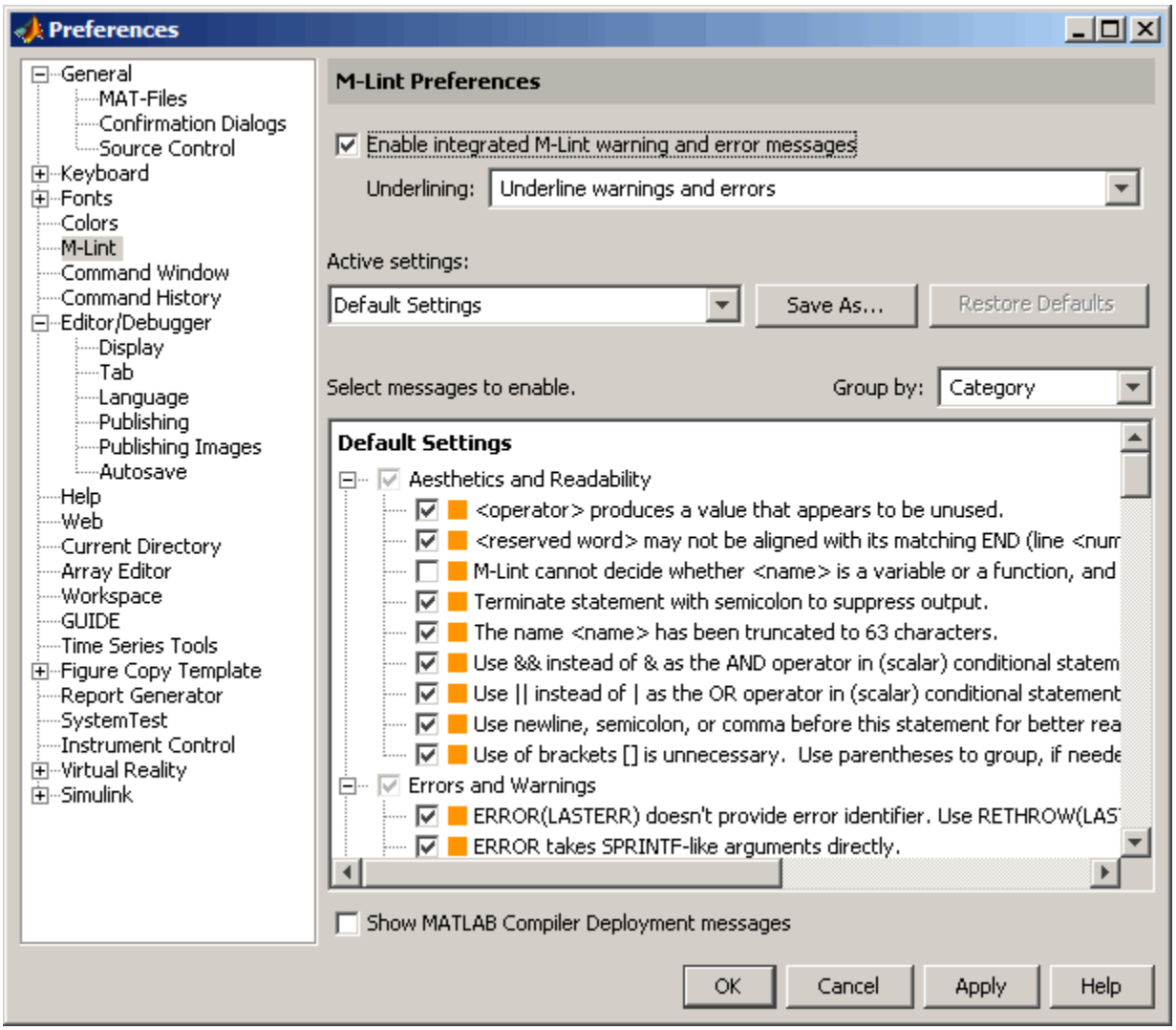
Desktop

New features and changes introduced in Version 7.3 (R2006b) are described here.

M-Lint Preferences Added and Now Appear on M-Lint Panel

M-Lint preferences now appear on a new M-Lint panel. There are new M-Lint preferences to disable specific messages or categories of messages, as well as to save the settings for use in a later session. In addition, you can choose to show or hide messages for the MATLAB Compiler product if the MATLAB Compiler product is installed. These new M-Lint preferences apply to the M-Lint code

analyzer operating automatically in the Editor/Debugger as well as to the M-Lint Code Check Report you run via the Current Directory browser Directory Reports or **Tools > M-Lint > Show M-Lint Report**.



Compatibility Considerations

M-Lint preferences were previously accessed via the Editor/Debugger Language settings for M.

Close All Documents and Close Selected Documents Feature Added

When you have multiple documents open within a tool, such as M-files in the Editor/Debugger, select **File > Close** to readily close selected files in that tool. Alternatively, right-click the document bar to close all open documents or all open documents except the selected document in that tool.

Accessibility Documentation Included

Accessibility features and assistive technologies are now part of the Desktop documentation. In prior versions, they were documented in the general Release Notes.

New Look and Feel on Linux and Solaris Platforms

MATLAB has a new look and feel on the Linux operating system from Linus Torvalds and the Sun Microsystems Solaris operating system that affects windows, decorations, color schemes, and responsiveness of the interface in MATLAB.

Compatibility Considerations

All of the changes are cosmetic, except for file dialog boxes, like Open file. The new file dialog boxes are more conventional and easier to use than in previous versions, and there is no loss in functionality.

Invalid info.xml File on Path Now Generates an Error

When MATLAB finds an `info.xml` file on the search path or in the current directory, it assumes the file is intended to add information to the **Start** button or the Help browser, and automatically validates the file against the supported schema. If there is an invalid construct in the `info.xml` file, MATLAB displays an error in the Command Window. The error is typically of the form

```
XML-file failed validation against schema located in
...
XML-file name: full path to...\info.xml
```

and might appear when you start MATLAB, press the **Start** button, or in other situations. For more information about the error and how to eliminate it, see Add Your Own Toolboxes to the Start Button.

Compatibility Considerations

In previous versions, MATLAB displayed a warning when it encountered an invalid `info.xml` file.

Help

New features and changes introduced in Version 7.3 (R2006b) are described here.

Exact Phrase and Wildcard Searching Added; Change to Search Database

These improvements were made to the Help browser search feature. Note that they are not supported on Japanese systems.

- Search Field Always Shown — The **Search for** field is now always in view when the Help browser is open. The list of pages found appears in the **Search Results** tab.
- Exact Phrase Searches for More Relevant Results — Find an exact phrase by typing quotation marks around the search term. For example, `"plot tools"` finds only pages that include `plot tools` together, but does not find pages that include `plot` in one part of the page and `tools` in another part of the page. You can specify more than one exact phrase in a search term, such as `"plot tools" "figure pallette"`.
- Wildcards (*) in Search Terms for Variations of a Word (Partial Word Search) — Use the wildcard character (*) in place of letters or digits in your search terms. For example, `plot*` finds various forms of the word `plot`, such as `plot`, `plots`, and `plotting`, while `p*t` find those variations as well as variations of `print` and `part`, among others. You can use multiple wildcards in a word or search term.

- Boolean Operator Evaluation Order Changed — Boolean `NOT` operators in search terms are now evaluated first, followed by `OR`s, and then `AND`s. In prior versions, Boolean operators were evaluated in left to right order.

Compatibility Considerations

The search enhancements were facilitated by changing to a new type of database. As a result, any existing Help search databases for non-Mathworks products will not work in R2006b and beyond. The documentation still displays in the Help browser, but it is not included in searches.

If you use Help browser documentation for non-MathWorks products with a pre-R2006b search database, a message will display in the Help browser to notify you that the documentation will not be included in searches. If you want search to work for that documentation, contact the product provider to request an R2006b-compatible search database.

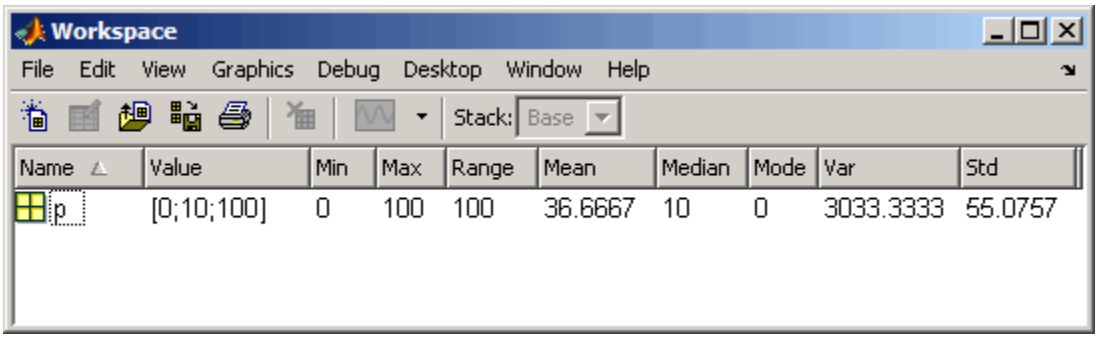
If you provide documentation for the Help browser and want the documentation to be included in the searches, you need to update the `helpsearch.db` entry in the `info.xml` file to the `helpsearch` directory, and prepare an R2006b-compatible help search database. This process is much easier than in the past. For instructions, contact Technical Support.

Workspace, Search Path, and File Operations

New features and changes introduced in Version 7.3 (R2006b) are described here.

Statistical Results in Workspace Browser

The Workspace browser includes new columns that automatically display results of common statistical calculations: Minimum, Maximum, Range, Mean, Median, Mode, Variance, and Standard Deviation. Use **View > Choose Columns** to specify the columns to show.



See also **File > Preferences > Workspace** for associated preferences.

Open Larger Arrays in Array Editor

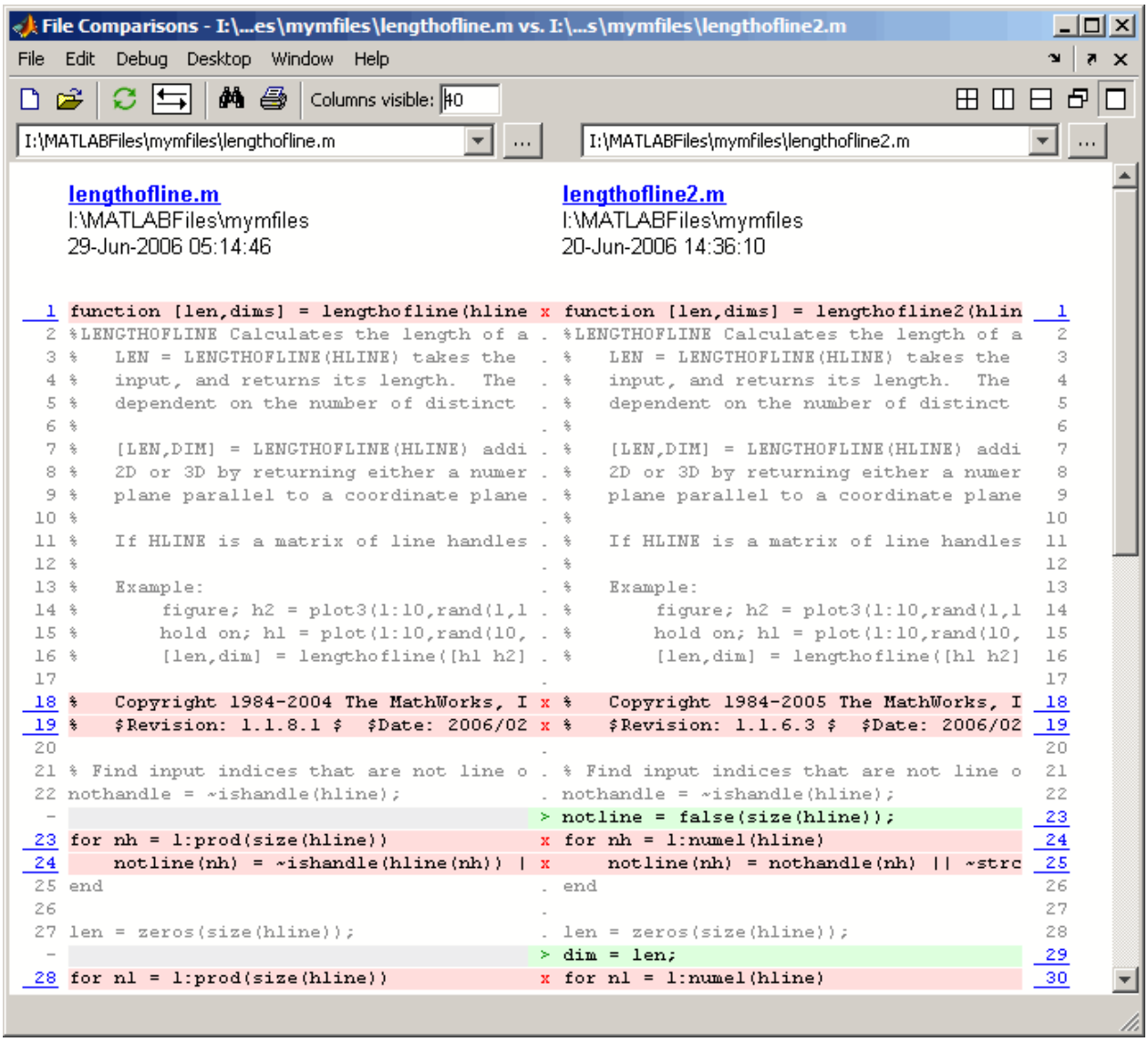
You can open larger arrays in the Array Editor. In previous versions, large arrays would fail to open in the Array Editor.

Editing and Debugging M-Files

New features and changes introduced in Version 7.3 (R2006b) are

File Comparisons Tool Added

Use the new File Comparisons tool to highlight the differences between two files. Open a file, select **Tools > Compare Against**, and then browse to select the second file or drag it into the tool from the Current Directory browser or Windows Explorer. The two files appear aligned in a window with highlights and indicators for any lines that differ, as shown in this example. For more information, see Comparing Files and Folders.



Use elements in the toolbar to exchange the left-right positions of the two files and to specify the number of columns shown.

You can also run the File Comparisons tool from the desktop by selecting **Desktop > File Comparisons**. Drag a file from the Current Directory browser or Windows Explorer into the tool. Then drag the second file into the tool.

M-Lint Enhancements for Suppressing Messages, and Showing Messages for the MATLAB Compiler Product

These are the new features and changes to M-Lint:

- "General Enhancements—Japanese Messages, and Line Number in Indicator Bar" on page 32-7
- "Suppressing M-Lint Messages" on page 32-7
- "Deployment Messages in M-Lint" on page 32-7

General Enhancements—Japanese Messages, and Line Number in Indicator Bar

- On Japanese systems, messages now appear in Japanese.
- The messages that appear at the indicator bar now includes the line number.

Suppressing M-Lint Messages

The M-Lint automatic code analyzer in the Editor/Debugger now provides a few ways for you to suppress specific messages and categories of messages:

- Ignore Specific Instance — Right-click at the underline for an M-Lint message, and from the context menu, suppress just that instance. M-Lint adds `%#ok` and a new message ID tag to the end of the line, which is the syntax in MATLAB for suppressing an M-Lint message.
- Disable All Instances — Right-click at the underline for an M-Lint message, and from the context menu, disable all instances of that message in all files. This changes the preference setting for that message.
- Use Preferences to Disable All Instances — Select **File > Preferences > M-Lint**, and disable specific messages or categories of messages, which applies to all instances in all files. You can save the settings to an M-Lint preference file. From the Editor/Debugger, you can select the M-Lint preference file from the **Tools > M-Lint** menu.

The M-Lint Code Check report also uses the preferences and suppresses the display of specific messages and categories of messages, per the preference settings.

Deployment Messages in M-Lint

M-Lint now displays deployment messages for the MATLAB Compiler product, such as `MCC does not permit the CD function`, which appear if the MATLAB Compiler product is installed and you select the M-Lint preference to **Show MATLAB Compiler deployment messages**. You can also disable or enable all deployment messages via the Editor/Debugger **Tools > M-Lint** menu. You can suppress specific messages using the same methods as for other M-Lint messages.

Compatibility Considerations

M-Lint preferences were previously accessed via the Editor/Debugger Language settings for M. See also "mlint Message IDs Changed and %#ok Syntax Enhanced" on page 32-8

Restore Breakpoints Using New dbstop Feature

You can save the status of breakpoints to a MAT-file using `s=dbstatus` and restore the breakpoint status at a later time by loading the MAT-file using the new `dbstop(s)` option. For example, set

breakpoints in `collatz`. Run `s=dbstatus` to assign the breakpoint status to `s`; use the `'completenames'` option to save absolute pathnames and breakpoint function nesting sequence. Save `s` to a MAT-file by running, for example, `save mydebugsession s`. At a later time, run `load mydebugsession` to restore `s`, and then run `dbstop(s)` to restore the breakpoints.

End Debugging Completely Using New dbquit('all') Option

If you debug multiple files at once, you can exit debug mode for all files by running `dbquit('all')`. If you debug `file1` and step into `file2`, running `dbquit` terminates debugging for both files. However, if you debug `file3` and also debug `file4`, running `dbquit` terminates debugging for `file4`, but `file3` remains in debug mode until you run `dbquit` again. The new `dbquit('all')` option ends debugging for both `file3` and `file4`, and as `dbquit` does, ends debugging for `file1` and `file2`.

Tuning and Managing M-Files

New features and changes introduced in Version 7.3 (R2006b) are described here.

File Comparison Directory Report Removed; Replaced by File Comparison Tool

The File Comparison Report, one of the directory reports available in the Current Directory browser, was removed.

Compatibility Considerations

The new File Comparisons tool replaces the File Comparison Report. The tool provides the same functionality as the report did, and adds new features. For details, see “File Comparisons Tool Added” on page 32-6.

M-Lint Code Check Report Enhancements and Changes

You can indicate specific messages or categories of messages you want M-Lint to report. For details, see “M-Lint Enhancements for Suppressing Messages, and Showing Messages for the MATLAB Compiler Product” on page 32-7.

Compatibility Considerations

In previous versions, the M-Lint Code Check report showed all messages except those suppressed via a `%#ok` at the end of a line. Now, the M-Lint Code Check report will show only the messages that are enabled via M-Lint preferences.

mlint Message IDs Changed and %#ok Syntax Enhanced

The `mlint` function with the `-id` option returns message IDs using a new form. For example, when you run

```
mlint('filename.m', '-id')
```

MATLAB Version 7.2 (R2006a) returns

```
L 22 (C 1-9) 2:AssignmentNotUsed : The value ...
```

whereas MATLAB Version 7.3 (R2006b) returns

```
L 22 (C 1-9): NASGU: The value ...
```

The `%#ok` syntax used in M-files to suppress the display of M-Lint messages for a line has been enhanced to support multiple messages per line. For example, given this line in an M-file,

```
data{nd} = getfield(flds,fdata{nd});
```

two M-Lint messages result:

```
34: 'data' might be growing inside a loop; ...
            consider preallocating for speed.

34: Use dynamic fieldnames with structures instead of GETFIELD...
           Type 'doc struct' for more information.
```

In MATLAB Version 7.2 (R2006a), M-Lint messages could only be suppressed for the entire line:

```
data{nd} = getfield(flds,fdata{nd}); %#ok
```

In MATLAB Version 7.3 (R2006b), you can still use `%#ok` to suppress all messages for the line, or you can add an ID tag to indicate the exact messages to suppress. For example, this suppresses only the first message about `data` growing inside a loop:

```
data{nd} = getfield(flds,fdata{nd}); %#ok<GFLD>
```

To suppress more than one message per line, separate the ID tags with a comma. For example, this suppresses both messages:

```
data{nd} = getfield(flds,fdata{nd}); %#ok<GFLD,AGROW>
```

Compatibility Considerations

In previous versions, the message IDs returned from `mlint` with the `-id` option, were of a form that included a numeric identifier, followed by the category. If you rely on the exact ID values, you will need to make modifications to use the new form.

If you use M-Lint in MATLAB Version 7.3 (R2006b) and then run those files a previous version, M-Lint in the previous version ignores the tag and IDs that follow the `%#ok`, and therefore suppresses all messages for that line, which is the expected behavior for the previous version.

mlintrpt Option Added to Use Preference Settings File

The `mlintrpt` function now accepts a new option that applies the preferences saved to an M-Lint settings file. It works with the `file` or `dir` syntax:

```
mlintrpt('fullpath_to_file', 'file', 'fullpath_to_configfile.txt')
mlintrpt('fullpath_to_dir', 'dir', 'fullpath_to_configfile.txt')
```

For example, create the file `NoSemiSetting.txt` by saving settings in **File** > **Preferences** > **M-Lint**. Later, use the settings via preferences or M-Lint in the Editor/Debugger, or via `mlintrpt` as in this example:

```
mlintrpt('lengthofline.m', 'file', 'I:\MATLABFiles\NoSemiSettings.txt')
```

which displays an M-Lint report in the MATLAB Web browser for the `lengthofline` file in the current directory using the M-Lint settings in `I:\MATLABFiles\NoSemiSettings.txt`.

Toolbar Refresh Button Removed

The Profiler included a refresh button  on the toolbar. This button has been removed. It performed an action similar to the **Refresh** button that appears in many of the Profiler reports, so use that button instead.

Publishing Results

notebook Setup Arguments Removed

In MATLAB 7.1 (R14SP3), the `notebook` function was enhanced to automatically detect the version of the Microsoft Word application and other required information. This feature changed the `notebook -setup` syntax—arguments to specify the version of the Microsoft Word application (`wordversion`, `wordlocation`, `templatelocation`) were no longer supported. If you used those arguments, MATLAB ignored them and issued a warning. Now in MATLAB 7.3 (R2006b), if you use those arguments, MATLAB errors.

Compatibility Considerations

If you use notebook with the `wordversion`, `wordlocation`, and `templatelocation` arguments in any of your files (for example, `startup.m`), remove them to prevent errors.

Mathematics

New Functions

Function	Description
`amd`	Interface to the `amd` algorithm. This interface is similar to that used in `symamd`, but is typically faster than `symamd`.
`bvpxtend`	Forms a guess structure for extending the boundary value problem solution
`ldl`	Full `ldl` factorization and solving for Hermitian matrices

max and min Now Use Magnitudes and Phase Angle for Complex Input

For complex input, `min` and `max` are computed using the magnitude, `min(abs(x))` and `max(abs(x))` respectively. In the case of equal magnitude elements, the phase angle, `min(angle(x))` and `max(angle(x))`, is now used.

Compatibility Considerations

In previous versions, `min` and `max` were computed using the magnitude, and in the case of equal magnitude elements, the first element was used. This behavior was indeterministic and not consistent with the `sort` function.

Code in previous releases that relied on the output of this functionality for the case described above should be updated.

Additional Output from lu

The `lu` function returns an additional output that helps improve numerical stability of sparse `lu` factorization.

Upper and Lower Factors for chol and ldl

Upper and lower triangular factors for `chol` and `ldl` improve performance for sparse `chol`. Referencing the lower triangle and returning the lower triangular factor is more memory efficient in the case of sparse `chol`.

Permutation Vectors with lu, luinc, ldl

Permutation vectors for `lu`, `luinc`, and `ldl` provide memory savings and, for large data, a noticeable performance improvement. You can now store permutation information in a single 1-by-N vector instead of an n-by-n matrix,

Two-Element Threshold for lu, spparms

A new two-element threshold for `lu` and `spparms` gives you more control over sparse `lu` and sparse `\` behavior.

Lower Triangular Factors from symbfact

Extra arguments exported for `symbfact` allow the return of the lower triangular symbolic factor.

Support for New Versions of AMD, COLAMD, CHOLMOD, UMFPACK

MATLAB 7.3 supports new versions of the following libraries:

Library Name	Version Supported in MATLAB 7.3
AMD	2.0
COLAMD	2.5
CHOLMOD	1.1
UMFPACK	5.0

Sparse Arrays on 64-Bit Systems

The internal storage of sparse arrays on 64-bit systems has changed in the R2006b release. This change should be invisible to most MATLAB users. However, MEX-file programs that run on a 64-bit system and access sparse arrays must be modified and recompiled in order to run on MATLAB Version 7.3. This applies to existing MEX-files and to any new MEX-files that you write.

If you are affected by this change, you will need to take the following steps to make your code compatible with the new sparse array format:

- Modify all sparse array declaration statements in MEX-files that are to run on a 64–bit system so that they now use the `mwSize` and `mwIndex` types instead of more specific type declarations such as `int` or `INTEGER*4`.
- Change any MEX code that assigns a negative value to a variable used to hold a sparse array index or size. An example of this would be code that temporarily uses an array index as a flag, assigning a -1 to it to mark the index as unset. Do not try to cast negative `int` values to `mwSize` or `mwIndex`. Instead, change your code to avoid using negative values.
- Recompile these MEX-files using the `-largeArrayDims` option

For a full description of what you will need to do, please read the section “Sparse Arrays on 64-bit Systems” on page 32-26 in the MATLAB External Interfaces release notes.

Compatibility Considerations

Existing MEX-files that run on 64-bit systems and access sparse arrays in MATLAB will not operate correctly in MATLAB 7.3 unless the required changes outlined above are made and the files are recompiled with the `-largeArrayDims` option. New MEX-files that you create must also adhere to these guidelines.

FFTW Upgraded to Version 3.1.1 in MATLAB

The version of FFTW used in MATLAB has been upgraded from 3.0.2 to 3.1.1 Please read the “Compatibility Considerations” on page 32-13 section, below.

Other changes in MATLAB FFTW support are:

- The default planner method is now `estimate`. Previously, `hybrid` was the default.
- There are new syntaxes for importing from and exporting to the library's internal single- and double-precision wisdom databases. These are as follows and are documented in the reference page for the MATLAB `fftw` function:
  - `str = fftw('dwisdom')`
  - `str = fftw('swisdom')`
  - `fftw('dwisdom', str)`
  - `fftw('swisdom', str)`

Compatibility Considerations

FFTW version 3.1.1 does not support wisdom produced by previous versions of the FFTW library. For this reason, FFTW wisdom that has been exported from a previous version of MATLAB cannot be imported successfully in MATLAB R2006b. Trying to import “old” wisdom results in a warning.

Future Obsolete Functions

The `betacore` function generates a warning message in R2006b but completes successfully. This function will be made obsolete in a future release.

Compatibility Considerations

You should remove all instances of the `betacore` function from your M-file program code. This function will not be supported in a future release of MATLAB.

Obsolete Functions

The following uses of MATLAB functions are obsolete as of this release. Attempts to use this functionality in the R2006a release generate a warning message but complete successfully. In R2006b, these operations fail and generate an error message.

- The following functions in their entirety: `bvpval`, `quad8`, `table1`, `table8`, `bessela`
- The `sort` function, when used with complex integer inputs
- The `gt`, `ge`, `lt`, and `le` functions, when used with complex integer inputs
- The `beta` function, when used with 3 inputs
- The `gamma` function, when used with 2 inputs
- The `opts.cheb` and `opts.stagtol` fields in the options structure of `eigs`

Compatibility Considerations

You will need to remove all instances that reference these functions at this time. These functions are no longer supported in MATLAB.

max and min No Longer Return Warning Messages for Inputs with Different Data Types

In MATLAB version 7.0 (Release 14), the functions `max` and `min` were changed to return results of a different data type than in previous releases. This behavior is described in max and min Now Have Restrictions on Inputs of Different Data Types. This change in behavior produced warning messages to assist you with diagnosing any resulting issues. In R2006b, these warning messages no longer exist.

Compatibility Considerations

The warning messages for mixed-type inputs to the functions `max` and `min` are no longer produced. Turning warning messages on will no longer display messages for this behavior, and you will no longer be able to depend on the messages for the diagnosis of problems.

Data Analysis

Generate M-File Now Supports Basic Fitting and Data Statistics

The **Generate M-File** option on the **File** menu of MATLAB figure windows now generates code that reproduces plot objects created with the Basic Fitting Tool or the Data Statistics Tool. The generated M-file function accepts new data as input and creates plot objects with the same graphics properties as those in the generating figure. In addition, the M-file recomputes fits and statistics for the new data. The code shows you how to program what you do interactively with the Basic Fitting Tool or the Data Statistics Tool.

New Options for Working with Time Series Objects

- Time Series Tools are now easier to find. From the MATLAB **Start** button, select **MATLAB** > **Time Series Tools**.
- Time series objects are now fully supported by the Array Editor. When you open a `timeseries` object (using `open` or the Workspace browser), the editor from Time Series Tools appears in the Array Editor.
- Time series objects can now be opened directly in Time Series Tools. Select a `timeseries` object in the Workspace browser, right click, and choose **Open in Time Series Tools** from the context menu.
- In Time Series Tools, you can now change subplot indices interactively. Click on a plotted line in a time series view and drag and drop it from one subplot to another. To create a new subplot, drag and drop the plotted line below the bottom axes.

Programming

Saving to MAT-Files Larger than 2 GB

With MATLAB R2006b, you can save data items that are over 2 gigabytes in size. This capability is implemented in MATLAB using a new HDF5-based format for MAT-files.

To save to a MAT-file in this format, specify the new `-v7.3` option with the `save` function:

```
save -v7.3 myfile v1 v2
```

Note MATLAB Version 7.3 does not write MAT-files in HDF5 format by default in this release; you must explicitly specify the `-v7.3` switch.

In this release,

- The default MAT-file format employed by `save` is the standard MAT-file format used in the R14, R14SP, and R2006a releases. You can explicitly specify this format with the command `save -v7`.
- To write an HDF5-based MAT-file, you must use `save -v7.3`.
- HDF5-based MAT-files are *not* compressed.

In a future release,

- The default MAT-file format will be the HDF5-based format. You can explicitly specify the HDF5-based format with the command `save -v7.3`.
- To write the standard MAT-file format, you must use `save -v7`.
- As of release R2008a, HDF5-based MAT files are compressed.

Here is a summary of the version options that you can use with `save`:

save Option	Description	Available In Versions	Is the Default In Versions
`save -v4`	Save to a MAT-file you can open in MATLAB version 4	V5 and later	V4, V5
`save -v6`	Save to a MAT-file you can open in MATLAB versions 5 or 6	V7 and later	V6
`save -v7`	Save to a MAT-file you can open in MATLAB version 7	V7.3 and later	V7.3
`save -v7.3`	Save to a MAT-file that supports data items ≥2 GB	V7.3 and later	post V7.3

Compatibility Considerations

If you are running MATLAB on a 64-bit computer system and you attempt to `save` a variable that is too large for a version 7 (or earlier) MAT-file, that is, you save without using the `-v7.3` option, MATLAB skips that variable during the `save` operation and issues a warning message to that effect.

If you are running MATLAB on a 32-bit computer system and attempt to `load` a variable from a `-v7.3` MAT-file that is too large to fit in 32–bit address space, MATLAB skips that variable and issues a warning message to that effect.

Import Wizard Generates Import M-Code

The Import Wizard now includes an automated M-code generation option. Click the Generate M-code button located in the lower right of the Import Wizard window and MATLAB generates an M-file that you can use to import additional files that are similar in type to the file you are currently importing. This simplifies the process of importing multiple files into MATLAB.

New Dynamic Regular Expression Syntax

The new `(?@cmd)` operator specifies a MATLAB command that `regexp` or `regexp` is to run while parsing the overall match expression. Unlike the other dynamic expressions in MATLAB, this operator does not alter the contents of the expression it is used in. Instead, you can use this functionality to get MATLAB to report just what steps it's taking as it parses the contents of one of your regular expressions. This can be helpful in debugging regular expressions, for example.

New Reserved MATLAB Keywords

Two new reserved keywords have been added to the MATLAB language in this release:

- `parfor` – designates a `for` loop in the Distributing Computing Toolbox
- `classdef` – signals the beginning of a MATLAB class definition

The `iskeyword` function returns `true` for each of these functions, thus identifying them as reserved keywords in MATLAB:

Compatibility Considerations

MATLAB keywords are reserved for use internal to MATLAB, and should not be used in your own program code as identifiers or function names. If your code uses either of these two new keywords in this manner, you should modify your code and use words that are not reserved.

Enhancements to Display Generated by whos

The visual output of the information returned by `whos` looks slightly different in R2006b than in previous releases. Changes are as follows:

- There is a new column in the output called `Attributes` that identifies values that are `sparse`, `complex`, `global`, or `persistent`.
- The words “array” and “object” have been dropped from items under the `Class` heading. Items formerly listed as `double array` or `timer object` , for example, are now displayed as `double`, and `timer`.
- MATLAB no longer includes summary information showing the “Grand total” of elements and bytes at the bottom of the `whos` output display.
- There is an additional field in the structure returned by `whos`. This new field is `'persistent'` and is set to logical 1 for those variables that are persistent, or logical 0 otherwise.

Here is an example of the information displayed by whos in MATLAB 7.3:

```
whos
  Name            Size             Bytes  Class              Attributes

  vCell           5x7               2380  cell
  vComplex        6x2                192  double             complex
  vDouble         6x5x9             2160  double
  vFuncHdl        1x1                 16  function_handle
  vGlobal         0x0                  0  double             global
  vObject         1x1                248  timer
  vPersist        0x0                  0  double             persistent
  vSparse        15x15               244  double             sparse
  vStruct         1x3                804  struct
```

Compatibility Considerations

If any of your programs depend on the displayed output of `whos`, specifically in relation to the changes listed above, you might have to modify your program code. Also, if your code relies on a specific number of fields for the output structure, you should be aware that this release adds a new field: `persistent`.

unique Function Returns First and Last Indices

Using the new `first` option with the `unique` function returns a vector with elements that represent the *lowest* indices of unique elements in the input vector.

Given the following vector `A`

```
A = [16 7 5 41 2 16 8 2 6 3 16 6 2 5 2 5];
```

Get a sorted vector of the unique elements of `A`:

```
v = unique(A)
v =
     2     3     5     6     7     8    16    41
```

Use the `first` and `last` options to get indices of the first and last elements in `A` that make up the sorted vector `v`:

```
[v, m1] = unique(A, 'first');
m1
m1 =
     5    10     3     9     2     7     1     4

[v, m2] = unique(A, 'last');
m2
m2 =
    15    10    16    12     2     7    11     4
```

save Compression and Unicode Options Removed

In MATLAB Versions 7.0 through 7.2, you could use the switches `–compress`, `–nocompress`, `–unicode`, and `–nounicode` with the `save` function to enable or disable compression and Unicode character encoding in the MAT-file being generated. In MATLAB Version 7.3, the `save` function no longer supports these switches. Instead, `save` now creates a MAT-file with compression and Unicode character encoding by default, or with the following command:

```
save -v7 filename
```

You can still save to a MAT-file without using compression or Unicode encoding. In fact, you will have to do this in order to create MAT-files that you can load into a MATLAB Version 6 or earlier. To save to a MAT-file without compression or Unicode encoding, type

```
save -v6 filename
```

To disable compression and Unicode encoding for all `save` operations in a MATLAB session, open the MATLAB **Preferences** dialog, select **General** and then **MAT-Files**, and then click the button labelled **Ensure backward compatibility (-v6)**.

Compatibility Considerations

If you have code that uses any of these option switches with the `save` function, you will need to modify that code to use the `–v6` option instead.

Warning Generated by try-catch

To accommodate future changes in the MATLAB error handling capabilities, MathWorks has added a new restriction to the single-line syntax of the `try-catch` block. In this release, the following syntax operates as it did in previous releases, but now it also generates the following warning message:

```
try try_statements, catch catch_statements, end

Warning: This try-catch syntax will continue to work in R2006b,
but may be illegal or may mean something different in future
releases of MATLAB.
```

To make this single-line `try-catch` work without warning in R2006b, you must insert a separator character (comma, semicolon, or newline) immediately after the word `catch`

```
try try_statements, catch, catch_statements, end
```

As with previous releases, the recommended syntax for a `try-catch` block is as follows:

```
try
   try_statements
catch
   catch_statements
end
```

Compatibility Considerations

Your M-file programs may generate this warning if correct syntax for `try` and `catch` is not used.

Case-Sensitivity Warning Removed

The following warning has been removed from MATLAB in release R2006b:

```
"Function call foo invokes /somewhere/on/the/path/foo.m,
however, function /somewhere/ahead/on/the/path/FOO.m that
differs only in case precedes it on the path."
```

In previous versions of MATLAB, this warning message was triggered when you called a function such as `foo`, and all of the following were true:

- There was more than one MATLAB file of this name on the MATLAB path
- The names of these files differed only in letter case, and
- A MATLAB file of this name but with different case (e.g., `FOO.m`) preceded a file of matching case (e.g., `foo.m`) on the path

Earlier versions of MATLAB displayed this warning for the purpose of helping users cope with newly-introduced case sensitive dispatching changes. The warning is being removed at this time under the assumption that users are now sufficiently well-acquainted with the way MATLAB handles case sensitivity in function calls.

Compatibility Considerations

The dispatching behavior regarding to the case sensitivity is NOT changed with the removal of this warning message. If both an exact match and inexact match are present on the path, the exact match is always the one to be invoked.

fprintf(0,...) Now Throws an Error

Commands such as `fprintf(0, ...)` and `fwrite(0, ...)`, in which the file identifier is zero (the same as `stdin`) now result in an error being thrown. In previous releases, MATLAB did not throw an error in response to these commands, even though printing or writing to `stdin` is clearly not a valid option.

Compatibility Considerations

If any of your programs use lower-level MATLAB file I/O functions that send output to `stdin`, because these functions no longer ignore this type of operation, your code will now generate an error. You should modify your program code to use a file identifier other than zero.

Assigning Nonscalar Structure Array Fields to a Single Variable

In the R14 and R14 service pack releases of MATLAB, assigning a nonscalar structure array field to a single variable incorrectly resulted in an error. For example, in the following code, you should be able to assign `S.A` to one, two, three, or four output variables. However, if you assign to just a single variable, MATLAB throws an error:

```
% Create a 1-by-4 structure array S with field A.
S(1).A = 1;   S(2).A = 2;   S(3).A = 3;   S(4).A = 4;

% Assigning S(1).A and S(2).A works as expected.
[x y] = S.A
x =
     1
y =
     2

% Assigning only S(1).A should work, but does not.
x = S.A;
??? Illegal right hand side in assignment. Too many elements.
```

This has been fixed in MATLAB 7.3.

Compatibility Considerations

If any of your programs rely on this error being thrown, you will need to modify those programs.

Comma Separators Not Required in Function Declaration

As of Release 14, the function definition line in a function M-file no longer requires commas separating output variables. This now makes the function definition syntax the same as the syntax used when calling an M-file function:

```
function [A B C] = myfun(x, y)
```

Compatibility Considerations

This new syntax is not valid in MATLAB versions earlier than Release 14. When writing an M-file that you expect to run on versions both earlier and later than R14, be sure to separate any output variables in the function definition line with commas:

```
function [A, B, C] = myfun(x, y)
```

Improved Performance on Certain Platforms and Operations

In this release, MATLAB offers improved performance in the following areas:

- Improved performance on 64-bit Windows XP and Linux platforms. This is independent of the size of data set in use.
- Faster scalar indexing into cell arrays.
- Faster assignment of cell array data to variables.

Graphics and 3-D Visualization

Plotting Tools Are Now Modular Desktop Components

The three MATLAB plotting tools (Figure Palette, Property Editor, and Plot Browser) now function as desktop components like the Workspace Browser and the Array Editor. They dock, however, not to the MATLAB desktop but to a Figures window. Figures windows contain one or more figures, each of which is accessible by a tab. Turning on any plotting tool changes your figure group into a mini-desktop. You can undock, rearrange, tab, and resize the plot tools within the mini-desktop, and their state will persist across MATLAB invocations. There are just one set of plot tools for all your figures, but they only operate on figures contained in the group; undocked figures are free of the plotting tools until you redock them.

For more information see Plotting Tools – Interactive Plotting in the MATLAB Graphics documentation.

Version 6 Property Editor Has Been Removed

The MATLAB Version 6 Property Editor, one of the Plotting Tools, is no longer available. this means that the `'v6'` switch for the `propedit` function now produces an error instead of starting the version 6 property editor. The error message is

```
??? Error using ==> propedit
The Version 6 property editor is no longer available. Sorry!
```

Compatibility Considerations

If you have code that calls the version 6 property editor, you will need to modify it to use the modular plotting tools described above in “Plotting Tools Are Now Modular Desktop Components” on page 32-22. The `propedit` function remain otherwise the same.

New Desktop Printing GUI

Printing MATLAB figures has become easier as a result of combining the **Page Setup**, **Print Setup**, and **Print Preview** dialogs into one tabbed **Print Preview** dialog. You can now specify paper size, plot size and layout, color and line weight, header text, rendering, and other printing characteristics in this new dialog. The **Page Setup** and **Print Setup** dialogs still exist, and the **Print** dialog that you call from **File —> Print** remains the same. The **Page Setup** dialog is available on all platforms.

For more information, see Using Print Preview in the MATLAB Graphics documentation and `printpreview` in the MATLAB function reference documentation.

Compatibility Considerations

The **Page Setup** dialog no longer is available from the figure window **File** menu. However, it does continue to exist; you can raise it using the `pagesetupdlg` command. The old **Print Preview** dialog has been removed, however. The old **Print Setup** dialog can be raised using the command

```
print -dsetup
```

When you dismiss the **Print Setup** dialog with **OK**, the settings you made with it are saved, but nothing is printed at that time.

Zoom Mode Now Supports Mouse Scroll Wheel

If your mouse has a center scroll wheel, you can use it to zoom in and out of axes, as well as by clicking and/or dragging.

You can zoom in by positioning the mouse cursor where you want the center of the plot to be and either press the mouse button or rotate the mouse scroll wheel away from you (upward). Zoom out by positioning the mouse cursor where you want the center of the plot to be and either simultaneously press **Shift** and the mouse button, or rotate the mouse scroll wheel toward you (downward). Each mouse click or scroll wheel click zooms in or out by a factor of 2.

Data Cursor Text Can Now Be Programmatically Modified

You can now easily customize the text of datatips. The `datacursormode` function lets you specify the contents and formatting of text displayed by the data cursor tool. When the data cursor tool is active, you can use its context (right-click) menu to edit or specify the text update function that MATLAB executes to display datatips. For more information, see Data Cursor — Displaying Data Values Interactively in the MATLAB Graphics documentation and `datacursormode` in the MATLAB Function Reference documentation.

Customizing Zoom, Pan, and Rotate3D Data Explore Modes

You can now customize the behavior of data explore modes by modifying the zoom, pan and rotate3d objects that are dereferenced as follows:

```
h = pan(figure_handle)
h = rotate3d(figure_handle)
h = zoom(figure_handle)
```

These syntaxes create *mode objects* that you can use to control the behaviors of the explore tools. Among the effects you can achieve using these explore mode objects are

- Allow, change, or inhibit a mode for a specified axes
- Create callbacks for pre- and post-buttonDown events
- Change callbacks dynamically

See `pan`, `rotate3d`, and `zoom` in the MATLAB Function Reference documentation for details and examples.

Compatibility Considerations

Using mode objects can cause a forward incompatibility. In prior releases, explore modes did not return an argument. Therefore, code such as the above examples that you write to take advantage of the new API will not run in MATLAB versions prior to R2006b. zoom, pan, and rotate3d code written for previous MATLAB versions, however, will run as before (there is no backward incompatibility).

Improvements to MATLAB Graphics Documentation

A new section in the Graphics User Guide, Types of MATLAB Plots, now includes a gallery of graphs that catalogs the kinds of plots that you can create using MATLAB graphics functions. There are two tables containing labeled icons, one for Two-Dimensional Plotting Functions, and one for Three-Dimensional Plotting Functions, classified by plot type. Clicking the function name above any thumbnail plot in the gallery opens the reference documentation for that function.

Reference pages for functions that create, edit, annotate, and save plots have been enhanced in several ways:

- Thumbnail figures at the top of plotting functions and related GUI reference pages illustrate what you see when you invoke these functions.
- *GUI Alternatives* sections above syntax descriptions describe how to invoke a function (or similar capability) from a GUI, and provide links to relevant topics in the Graphics and Desktop Tools User Guides.
- Revised printing documentation in user guides and reference pages (see “New Desktop Printing GUI” on page 32-22, above)
- Numerous corrections and clarifications of details in user guides and reference pages

Creating Graphical User Interfaces (GUIs)

Functions Are Now Obsolete

The following functions are obsolete in MATLAB 7.3 (R2006b): `axlimdlg`, `edtext`, `menubar`, `pagedlg`, `umtoggle`.

Compatibility Considerations

The functions shown in the following table will continue to work but their use will generate a warning message. As soon as possible, replace any occurrences you may have of these functions with the function(s) shown in the following table, if any, or with other suitable code.

Function	Replacement
`axlimdlg`	None
`edtext`	Set the `Editing` property of the `text` object.
`menubar`	Set the `MenuBar` property of the figure to `none`.
`pagedlg`	`pagesetupdlg`
`umtoggle`	Set the `Checked` property of the `uimenu` object.

Colored Buttons on Windows XP

Setting the background color of user interface control (uicontrol) push buttons and toggle buttons on Windows XP now results in flat, colored buttons.

Compatibility Considerations

Prior to this release, if you set the background color of a push button or toggle button to any color other than the factory color on Windows XP, the color displayed only as a border around the button. With this release, any such buttons will display as flat, colored buttons with a simple border.

Documentation Enhancement

The Creating GUIs Programmatically section of the documentation now contains information commensurate with information in the Creating GUIs with GUIDE section. It adds, in workflow order, information and many basic examples about:

- Adding components, menus, and toolbars to your GUI. Placing and aligning components.
- Designing for cross-platform compatibility.
- Initializing the GUI and creating callbacks.
- Callback examples for the different components.

External Interfaces/API

New Types for Declaring Size and Index Values

Version 7.3 (R2006b) defines two new types for API arguments and return values. These are

- `mwSize` — represents size values, such as array dimensions and number of elements.
- `mwIndex` — represents index values, such as indices into arrays.

Using these types in array declarations replaces more specific type declarations such as `int` or `INTEGER*4`. In general, using these types consistently in your C or Fortran source files can insulate your code from changes in the API implementation that might take place between different versions of MATLAB.

The `mwSize` and `mwIndex` types are required when working with functions that access sparse arrays on a 64-bit system. This is described in the release note on “Sparse Arrays on 64-bit Systems” on page 32-26, below.

Note In Fortran, `mwSize` and `mwIndex` are implemented as preprocessor macros. To use these types in Fortran code, add the declaration `#include "fintrf.h"` in your source file and build your MEX-files using the Fortran preprocessor.

Sparse Arrays on 64-bit Systems

The internal storage of sparse arrays on 64-bit systems has changed in the R2006b release. Due to this change, you will need to

- Change declaration statements in your source code so that you use the new types `mwSize` and `mwIndex` in place of specific type declarations such as `int` or `INTEGER*4`. This is described in the section “New Types for Declaring Size and Index Values” on page 32-26, above.
- Recompile all MEX-files that interact with sparse arrays using the new `-largeArrayDims` switch. For more information about the `-largeArrayDims` switch, see the section “New MEX Switch” on page 32-26, below.

See the section on “Compatibility Considerations” on page 32-27 to find out if you will need to make any modifications to your existing MEX code to accommodate these changes.

Sparse API Functions Affected By This Change

You will need to recompile any MEX-files that use the following sparse functions on a 64-bit system:

- `mxGetIr`
- `mxGetJc`
- `mxSetIr`
- `mxSetJc`

New MEX Switch

In order to build MEX-files that use any of the sparse array functions listed above, you need to compile these files with the `-largeArrayDims` switch, as shown here:

```
mex -largeArrayDims filename
```

Also, any existing MEX-files that interact with sparse arrays in MATLAB Version 7.3 must be recompiled using the `-largeArrayDims` switch.

Note The `-largeArrayDims` option is likely to become the default in a future version of MATLAB.

Compatibility Considerations

If you are using any of the functions listed above, then you should be aware of the following potential compatibility issues if your MEX code uses sparse arrays on a 64-bit system:

- In release R2006b, you must rebuild all MEX-files that use sparse arrays using the new `-largeArrayDims` switch.
- Before building your MEX-files, change your C or Fortran sources to use the `mwSize` or `mwIndex` types introduced in this release. See the `mxArray` reference pages for the types to use for each function.
- MEX-files that compiled properly in Version 7.2 (R2006a) and do not use sparse arrays should build and execute correctly in Version 7.3 (R2006b) without changes.
- For more information on how the sparse API is affected, see the `Sparse Arrays on 64-Bit Systems` section in the MATLAB Mathematics release notes.

New MAT-File Format Based on HDF5

In Version 7.3 (R2006b), you can save MAT-files in a format based on HDF5. Unlike earlier MAT-file formats, the HDF5-based format is capable of saving variables that occupy more than 2 GB of storage, including large arrays created on 64-bit systems.

To save a MAT-file in the HDF5-based format, use the `-v7.3` option to the MATLAB `save` function or the `"w7.3"` mode argument to the C or Fortran `matOpen` function. The default MAT-file format is the same as that in Version 7.2 (R2006a).

Compatibility Considerations

Earlier versions of MATLAB cannot read MAT-files written in the HDF5-based format.

MAT-files written with MATLAB Version 7.3 (R2006b) on a 64-bit system can be read back into MATLAB 7.3 on a 32-bit system, provided that none of the values stored in the MAT-file require more than 32 bits to store.

-V5 Option to MEX to Be Removed

The `-V5` option to `mex` is not supported in this and future versions of MATLAB.

Compatibility Considerations

You will no longer be able to build a MEX-file that is compatible with MATLAB Version 5.

Location of mex.bat File Changed

In MATLAB Version 7.3, the Microsoft Windows script `mex.bat` is located in the directory `$MATLAB \bin`.

Compatibility Considerations

You may need to change any scripts or environment variables that relied on the previous location of `mex.bat`.

Changes to Compiler Support

Compaq Visual Fortran version 6.1 is supported in Version 7.3 (R2006b) but will not be supported in a future version of MATLAB.

Compatibility Considerations

To ensure continued support for building your Fortran programs, consider upgrading to another supported compiler. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Actxcontrol Command Now Validates ProgID

Attempting to insert a COM server into a MATLAB figure can result in unpredictable behaviors. To prevent this condition, the `actxcontrol` command now checks the ProgID by looking at the registry's `HKR/CLSID/Control` keyword and throws an error if the ProgID does not belong to a Microsoft ActiveX control.

Compatibility Considerations

Before the validation check was added, some server ProgIDs worked with `actxcontrol` , probably because there are cases where Microsoft software points the server GUID to a control GUID underneath. An example of a ProgID that might not work with `actxcontrol` is the Windows Media® Player server ProgID `Mediaplayer.mediaplayer.1`. Therefore, depending on how Microsoft software registers the control, the following command might now return an error:

```
h = actxcontrol('Mediaplayer.mediaplayer.1'); % Returns an error
```

The correct ProgID to use for Mediaplayer is the ActiveX control ProgID:

```
h = actxcontrol('WMPlayer.OCX.7'); % Use control ProgID
```

Note that methods and properties to open and play files are different when using the control ProgID.

You can disable the validation check with the following command:

```
feature('COM_ActxProgidCheck',0)
```

Or reenable it with:

```
feature('COM_ActxProgidCheck',1)
```

By default, ProgID checking is on.

R2006a

Version: 7.2

New Features

Compatibility Considerations

Desktop Tools and Development Environment

Startup and Shutdown

New features and changes introduced in Version 7.2 (R2006a) are described here.

Installation Directory Structure on Windows Platforms

The installation directory structure on Microsoft Windows platforms is slightly different than in previous versions. By default, the structure now includes a general `MATLAB` top level directory, with a subdirectory for `R2006a`. The root directory for the MATLAB software returned by the `matlabroot` function, is now of the form in this example:

```
D:\Applications\MATLAB\R2006a
```

In previous versions, the top-level directory included the version number, so the root directory for MATLAB, as returned by the `matlabroot` function, was of the form in this example:

```
D:\Applications\MATLAB 7.1
```

Compatibility Considerations

If you relied on the explicit root directory structure for MATLAB in your code, change it to reflect the new structure including the top-level `MATLAB` directory. The `matlabroot` function might be useful.

Error Log Reporting

If MATLAB experiences a segmentation violation, it generates an error log. Upon the next startup, MATLAB prompts you to e-mail the error log to The MathWorks. The MathWorks uses the log to work on resolving the problem. When you send a log, you receive a confirmation e-mail and will only receive further e-mails if The MathWorks develops a fix or workaround for the problem.

There are some situations where the Error Log Reporter does not open, for example, when you start MATLAB with a `-r` option or run in deployed mode. If you experience segmentation violations but do not see the Error Log Reporter on subsequent startups, you can instead e-mail logs by following the instructions at the end of the segmentation violation message in the Command Window.

JVM Software Updated for 64-Bit Linux Platforms

The Sun Microsystems JVM software version that MATLAB uses is now version 1.5.0_04 for 64-bit platforms running the Linux operating system from Linus Torvalds.

Desktop

New features and changes introduced in Version 7.2 (R2006a) are described here.

Preferences Reorganized and New Keyboard Pane Added to Support Command Window and Editor/Debugger

Preferences includes a new pane, **Keyboard**, for setting key bindings, tab completion, and delimiter-matching preferences for the Command Window and Editor/Debugger. Most of these preferences were previously located in the preference panes for the Command Window or Editor/Debugger.

Compatibility Considerations

You no longer access keyboard and indenting preferences for the Command Window and Editor/Debugger from the component preferences, but rather from the new **Keyboard** preferences. In addition, some preferences that were set separately for these components are now shared. For details about the changes, see “Keyboard and Indenting Command Window Preferences Reorganized” on page 33-3, and “Keyboard and Indenting Editor/Debugger Preferences Reorganized” on page 33-5.

Open All Desktop Tools from Desktop Menu

You can now open (and close) all desktop tools from the **Desktop** menu. In previous versions, you could not access document-based tools from the **Desktop** menu. The document-based desktop tools are: Editor/Debugger, Figures, Array Editor, and Web Browser.

Access Login Renamed to MathWorks Account

Use **Help > Web Resources > MathWorks Account** menu items to go to your MathWorks Account if you are registered, or to register online. MathWorks Account was previously called Access Login.

Running Functions — Command Window and Command History

New features and changes introduced in Version 7.2 (R2006a) are described here.

Keyboard and Indenting Command Window Preferences Reorganized

The Command Window Keyboard and Indenting preferences pane was removed. The tab size preference is now on the Command Window preferences pane. The tab completion, keybinding, and parentheses matching preferences were moved to the new Keyboard preferences pane. The parentheses-matching preferences are now called delimiter-matching preferences and are shared with the Editor/Debugger.

Help

New features and changes introduced in Version 7.2 (R2006a) are described here.

help for Model Files

You can now use the `help` function to get the complete description for MDL-files. For example, run

```
help f14_dap.mdl
```

MATLAB displays the description of the F-14 Digital Autopilot High Angle of Attack Mode model in the Simulink software, as defined in its **Model Properties > Description**:

```
Multirate digital pitch loop control for F-14 control design
demonstration.
```

Workspace, Search Path, and File Operations

New features and changes introduced in Version 7.2 (R2006a) are described here.

toolboxdir function added

The `toolboxdir` function returns the absolute pathname to the specified toolbox. It is particularly useful with the MATLAB Compiler product because the toolbox root directory is different than in MATLAB.

Visual Directory View Removed

The Visual Directory view was removed from the Current Directory browser. Most of the features it provided are accessible from the Current Directory browser standard view.

Editing and Debugging M-Files

New features and changes introduced in Version 7.2 (R2006a) are

Tab Completion — Tab Now Completes Function and Variable Names

You can now use tab completion in the Editor/Debugger to complete function names and variable names that are in the current workspace. When you type the first few characters of a function or variable name and press the **Tab** key, the Editor/Debugger displays a list of all function and variable names that begin with those letters, from which you choose one.

It operates essentially the same way as the existing tab completion feature in the Command Window, with the exception that Editor/Debugger tab completion does not support completion of file and path names.

To enable tab completion in the Editor/Debugger, select **File > Preferences > Keyboard**, and then under **Tab completion**, select **Tab key narrows completions**. By default, the preference is selected.

With tab completion enabled in the Editor/Debugger, you can still include tab spacing, for example, to include a comment at the end of a line. To add tab spacing, include a space after the last character you type and then press **Tab**. Instead of showing possible completions, the Editor/Debugger moves the cursor to the right where you can continue typing.

Compatibility Considerations

If you press the **Tab** key to add spacing within your statements, you might instead get a completion for a function or see a list of possible completions. For example, if the preference for tab completion is on and you want to create this statement

```
if a=mate     %test input value
```

where you press **Tab** after `mate` to achieve the spacing, the following happens instead

```
if a=material
```

This is because the tab completion preference completes `mate`, automatically supplying the `material` function.

To achieve the spacing with **Tab** (as in previous versions), either add a space after `mate` and then press **Tab**, or turn off the preference Tab key narrows completions in Keyboard Preferences.

Go Menu Added; Bookmark and Go To Items Moved from Edit Menu to Go Menu

- To set, clear, and navigate to bookmarks, use the menu items in the new **Go** menu, which were previously located in the **Edit** menu.
- The **Go To** feature for navigating to line numbers, functions in M-files, and cells has moved to the new **Go** menu. It was previously located in the **Edit** menu.

Compatibility Considerations

Use the new **Go** menu items instead of **Edit > Bookmark** features and **Edit > Go To**.

Navigate Back and Forward in Files

Use **Go > Back** (and **Go > Forward**) to go to lines you previously edited or navigated to in a file, in the sequence you accessed them. The main benefit of this feature is going directly to lines of interest. As an alternative to the menu items, use the Back and Forward buttons on the toolbar.

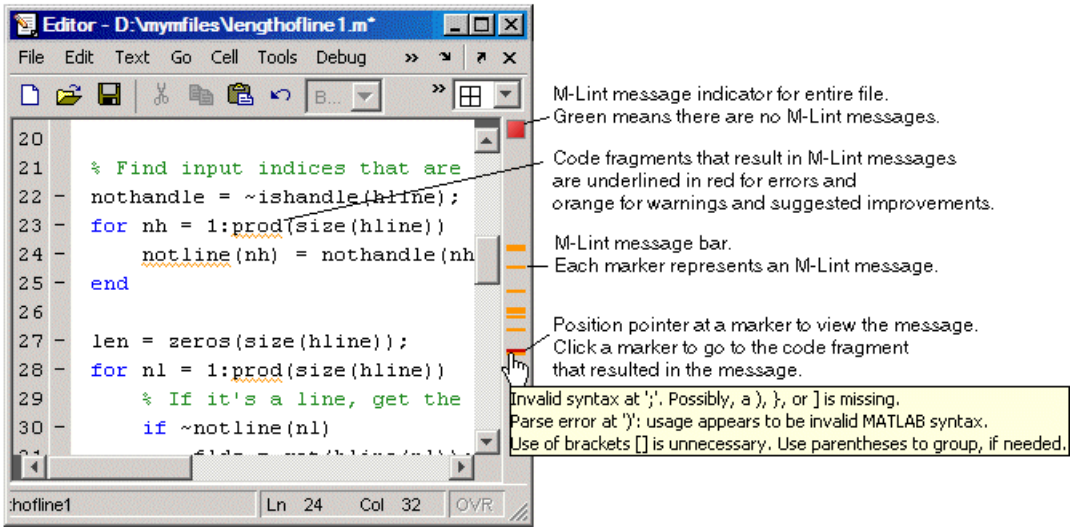
Keyboard and Indenting Editor/Debugger Preferences Reorganized

The Editor/Debugger **Keyboard and Indenting** preferences pane was renamed to **Tab** preferences, and keybinding and parentheses-matching preferences were moved to the new **Keyboard** preferences pane. The parentheses-matching preferences are now called delimiter-matching preferences and are shared with the Command Window.

M-Lint Automatic Code Analyzer Checks for Problems and Suggests Improvements

The M-Lint code analyzer, now built into the Editor/Debugger, continuously checks your code for problems and recommends modifications to maximize performance and maintainability. It performs the same analysis as the existing M-Lint Code Check report, but also provides these features:

- Indicates the problem lines and associated M-Lint messages directly in the M-file rather than in a separate report.
- Identifies (underlines) code fragments within a line that result in M-Lint messages.
- Distinguishes messages that report errors (red) from warnings and suggestions (orange).
- Continually analyzes and updates messages as your work so you can see the effect of your changes without having to save the M-file or rerun an M-Lint report.



To use or turn off M-Lint in the Editor/Debugger, select **File > Preferences > Editor/Debugger > Language**, and for **Language**, select `M`. Under **Syntax**, select **Enable M-Lint messages**, or clear the check box to turn it off. Use the associated drop-down list to specify the types of code fragments that you want M-Lint to underline, for example, **Underline warnings and errors**.

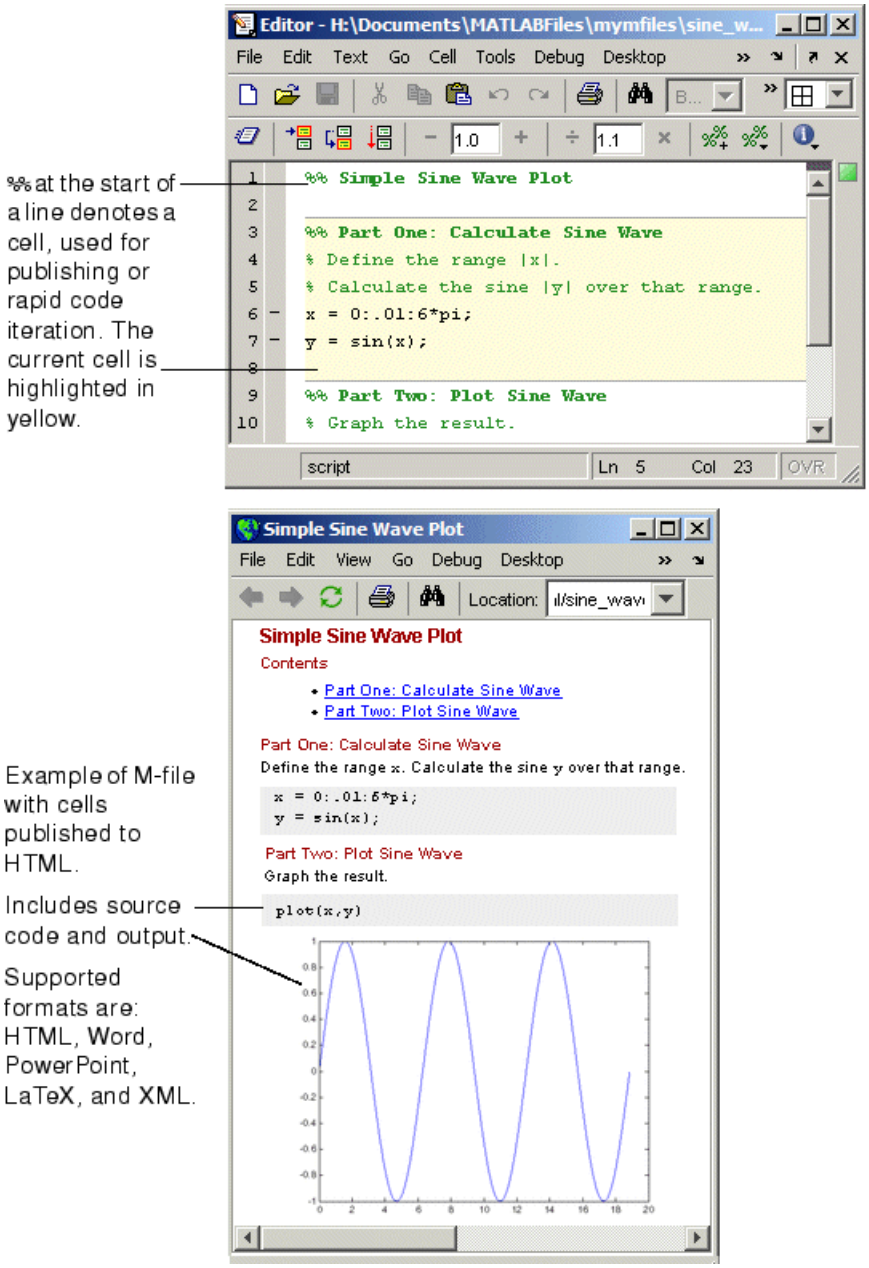
Debugging Changes

- The `dbstop` function now allows you to stop at, (not in), a non M-file, allowing you to view code and variables near it in your M-file. For example, if you want to stop at the point in your M-file `myfile.m` where the built-in `clear` function is called, run `dbstop in clear; mymfile`. Use this feature with caution because the debugger stops in M-files it uses for running and debugging if they contain the non M-file, and then some debugging features do not operate as expected, such as typing `help functionname` at the `K>>` prompt.

Cell Mode On by Default — Shows Cell Toolbar and Possibly Horizontal Lines and Yellow Highlighting; Cell Information Bar and Button Added

Cell mode, a useful feature in the Editor/Debugger for publishing results and rapid code iteration, is now enabled by default. An M-file cell is denoted by a %% at the start of a line. Any M-file that contains %% at the start of a line is interpreted as including cells. The Editor/Debugger reflects the cell toolbar state and the cell display preferences, such as yellow highlighting of the current cell and gray horizontal lines between cells.

For quick access to information about using cells in M-files, use the new information button on the cell toolbar.



If you do not want cell mode enabled, select **Cell > Disable Cell Mode**.

MATLAB remembers the cell mode between sessions. If cell mode is disabled when you quit MATLAB, it will be disabled the next time you start MATLAB, and the converse is true.

In MATLAB Version 7.2, the first time you open an M-file in the Editor/Debugger, the cell toolbar appears. If the M-file contains a line beginning with %%, an information bar appears below the cell toolbar, providing links for details about cell mode. To dismiss the information bar, click the close box on the right side of the bar. To hide the cell toolbar, right-click the toolbar and select **Cell Toolbar** from the context menu.

Compatibility Considerations

In previous versions, cell mode was off by default. The cell toolbar and yellow highlighting or horizontal rules in M-files that contain %% at the start of a line might be unexpected. If you used the %% symbols at the start of a line in M-files for a purpose other than denoting M-file cells, consider replacing the %% symbols with a different indicator, or keep cell mode disabled.

Lines Between Cells

You can set an Editor/Debugger display preference, **Show lines between cells**, to add a faint gray rule above each cell in an M-file. The line does not print or appear in the published M-file.

Cell Titles in Bold Preference Removed

Previous versions included an Editor/Debugger display preference to **Show bold cell titles**. When cleared, cell titles appeared in plain text, rather than bold text. This is no longer a preference you can set — all cell titles now appear in bold text.

Tuning and Managing M-Files

New features and changes introduced in Version 7.2 (R2006a) are

- “M-Lint and mlint Enhancements and Changes” on page 33-8
- “Profiling Enhancements” on page 33-9
- “Visual Directory View Removed from Current Directory Browser” on page 33-9

M-Lint and mlint Enhancements and Changes

The M-Lint code analyzer is now built into the Editor/Debugger where it continuously checks your code for problems and recommends modifications to maximize performance and maintainability. For details, see “M-Lint Automatic Code Analyzer Checks for Problems and Suggests Improvements” on page 33-5.

Compatibility Considerations

The `mlint` function has changed slightly to support its use in the Editor/Debugger. Specifically, the results returned from `mlint` with the `-id` option are of a different form than for previous versions. If you rely on the exact values, you need to make modifications.

For example, this is the form of a message returned in R2006a: `L 22 (C 1-9) 2:AssignmentNotUsed : The value assigned here to variable 'nothandle' might never be used`.

This is the form of the message from R14SP3: `22 (C 1-9) InefficientUsage:AssignmentNotUsed : The value assigned here to variable 'nothandle' might never be used`.

There is now a numeric identifier, followed by the category, for example: `2:AssignmentNotUsed`.

If you do rely on the exact values, note that there have been very few changes to the message text itself. For example, both R14SP3 and R2006a use the same text: `The value assigned here to variable 'nothandle' might never be used`.

Because of improvements being made to `mlint`, the values returned using the `-id` option are expected to change in the next version as well, particularly the numeric identifier and category form. Do not rely on the exact values returned using `mlint` with the `-id` option or you will probably need to make modifications.

Profiling Enhancements

nohistory Option Added to profile Function

Use the new `profile -nohistory` option after having previously set the `-history` option to disable further recording of history (exact sequence of function calls). All other profiling statistics continue to accumulate.

Accuracy Improved

The Profiler provides more accurate accounting. The total time you see with the Profiler GUI now matches total wall clock time from when you started profiling until you stopped profiling. Overhead associated with the Profiler itself is now applied evenly.

Statistics for Recursive Functions

The `profile` function now gathers and reports time for recursive functions in the `FunctionTable`'s `TotalTime` for the function. In previous versions, `profile` attempted to break out `TotalRecursiveTime`, which was not always accounted for accurately. The value for `TotalRecursiveTime` in `FunctionTable` is no longer used.

This change is also reflected in the Profiler GUI reports.

PartialData Reported in Results, AcceleratorMessages Removed

The `FunctionTable` now includes the `PartialData` value. A value of `1` indicates the function was modified during profiling, for example, by being edited or cleared, so data was only collected up until the point it was modified.

In previous versions, `FunctionTable` included `AcceleratorMessages` although it was not used. `AcceleratorMessages` is no longer included.

Visual Directory View Removed from Current Directory Browser

The Visual Directory view was removed from the Current Directory browser.

Compatibility Considerations

Most of the features it provided are accessible from the Current Directory browser standard view.

Publishing Results

New features and changes introduced in Version 7.2 (R2006a) are described here.

Insert Italic Text Markup

You can now make designated text comments in cells appear italicized in the published output. Use **Cell > Insert Text Markup > Italic Text**, or use the equivalent markup symbols, underscores, as in `_SAMPLE ITALIC TEXT_`.

publish Function has New catchError Option

The `publish` function has a new `catchError` option that allows you to continue or stop publishing if the M-file contains an error.

Source Control Interface

New features and changes introduced in Version 7.2 (R2006a) are described here.

PVCS Source Control System Name Change

The PVCS® source control system (from Merant) now has a new name, ChangeMan® (from Serena®), and the source control interface in MATLAB on UNIX platforms reflects this change.

If you use the ChangeMan software on UNIX platforms, the `cmopts` value returned for it is `pvcs`. If you use PVCS software, select `ChangeMan` in the Source Control Preferences pane.

Compatibility Considerations

PVCS software users on UNIX platforms formerly selected `PVCS` in the Source Control Preferences pane. Now, PVCS software users select `ChangeMan` instead.

Mathematics

New Library CHOLMOD for Sparse Cholesky Factorization

For sparse matrices, MATLAB now uses CHOLMOD version 1.0 to compute the Cholesky factor. CHOLMOD is a set of routines offering improved performance in factorizing sparse symmetric positive definite matrices. See the function reference pages for `chol`, `spparms`, and `mldivide` for more information on how CHOLMOD is used by MATLAB.

New Solver for State-Dependent DDEs

In this release, MATLAB provides a second solver function, `ddesd`, in addition to the existing `dde23` function, for delay differential equations (DDEs). This new solver is for use on equations that have *general* delays. You supply a function in the input argument list that returns the vector of delays to be used by the solver. See the function reference page for `ddesd`, and Delay Differential Equations in the MATLAB Mathematics documentation for more information.

Upgrade to BLAS Libraries

MATLAB now uses new versions of the Basic Linear Algebra Subroutine (BLAS) libraries. For Intel processors on Windows and Linux platforms, MATLAB supports the Math Kernel Library (MKL) version 8.0.1. For AMD processors on Linux platforms, MATLAB uses the AMD Core Math Library (ACML) version 2.7.

New Function for Integer Division

The new `idivide` function provides division similar to `A./B` on integers except that fractional quotients are rounded to integers according to a specified rounding mode.

New Input to gallery Function

The `gallery` function has a new, optional input argument called `classname`. The `classname` input is a quoted string that must be either `'single'` or `'double'`. When you specify a `classname` argument in the call to `gallery`, MATLAB produces a matrix of that class.

Improved Algorithm for expm

The `expm` function now uses an improved algorithm to compute a matrix exponential. This algorithm often requires fewer matrix multiplications.

More Efficient condest for Sparse Matrices

The `condest` function handles sparse matrices more efficiently when estimating a 1-norm condition number.

accumarray Accepts Cell Vector Input

The `accumarray` function now accepts a cell vector as the `subs` input. This vector can have one or more elements, each element a vector of positive integers. All the vectors must have the same length. In this case, `subs` is treated as if the vectors formed columns of an index matrix.

Data Analysis

Data Analysis Collection Revised and Expanded

In this release, the MATLAB Data Analysis collection has been thoroughly revised to improve content organization and flow. In addition, most examples have been updated and streamlined.

Reference Pages for timeseries and tscollection Objects

Detailed reference pages are now available for `timeseries` and `tscollection` objects, properties, and methods. You can access these reference pages in the Help contents, under Data Analysis in the MATLAB Function Reference.

Text Files Can Be Imported In Time Series Tools

In Time Series Tools, you can now use the Import Wizard to import data from text files, such as `.csv`, `.dat`, and `.txt`.

Linux 64 Platform Fully Enabled for Time Series Tools

Time Series Tools is now fully enabled on the Linux 64 platform.

Compatibility Considerations

On the Linux 64 platform, you no longer need to manually enable the Time Series Tools feature before starting Time Series Tools (as in MATLAB 7.1).

Programming

Larger Data Sets with 64-Bit Windows XP

MATLAB support for Windows XP 64-bit edition enables you to handle much larger data sets. To learn more about memory allocation for arrays, see Memory Allocation.

Using avifile and movie2avi on Windows XP 64

Note You must change the compression setting if you use the `avifile` or `movie2avi` function on Windows XP 64.

MATLAB currently defaults to using Indeo codecs to compress video frames when using `avifile/addframe` or `movie2avi`. If you attempt to use `avifile` and `addframe`, or `movie2avi` on a Windows XP 64-bit platform without specifying the compression type, an error message appears indicating the codec was not found. Nondefault settings must be explicitly passed in when using these functions on Windows XP 64 because Microsoft does not provide Indeo codecs on this platform.

This issue does not affect 32-bit Windows XP installations.

Compatibility Considerations

To work around this issue, do the following:

1. Explicitly specify no compression when creating the `avifile` object or when calling `movie2avi`. Two examples of this are

   ```
   aviobj = avifile('myvideo.avi', 'compression', 'none');
   movie2avi(mov, 'myvideo.avi', 'compression', 'none');
   ```

2. Specify a codec for a compression that is installed. The ones that are included with Windows XP 64 are

   - IYUV — Intel YUV codec (c:\winnt\system32\iyuv_32.dll)
   - MRLE — Microsoft RLE codec (c:\winnt\system32\msrle32.dll)
   - MSVC — Microsoft Video 1 codec (c:\winnt\system32\msvidc32.dll)

For example, to use the Intel YUV codec, use the four-CC code:

```
aviobj = avifile('myvideo.avi', 'compression', 'IYUV');
```

Other codecs can be found at `http://fourcc.org`.

Note there are restrictions with some codecs. For example, some codecs can only be used with grayscale images.

Regular Expressions

MATLAB 7.2 introduces the following new features for regular expressions in MATLAB. For more information on these features, see Regular Expressions in the MATLAB Programming documentation.

New Features

- Dynamic regular expressions — You can now insert MATLAB expressions or commands into regular expressions or replacement strings. The dynamic part of the expression is then evaluated at runtime.
- Generating literals in expressions — Use the new `regexptranslate` function when you want any of the MATLAB regular expression functions to interpret a string containing metacharacters or wildcard characters literally.
- New parsing modes — Four matching modes (case-sensitive, single-line, multiline, and freespacing) extend the parsing capabilities of the MATLAB regular expression functions.
- Warnings display — Use the new `'warnings'` option with the regular expression functions to enable the display of warnings that are otherwise hidden.

Compatibility Considerations

Calling `regexp` or `regexpi` with the `'tokenExtents'` and `'once'` options specified now returns a `double` array instead of a cell array. You may need to change your code to accommodate the new return type.

Setting Environment Variables

Use the new `setenv` function to set the value of an environment variable belonging to the underlying operating system.

issorted Support for Cell Arrays

You can now use the `issorted` function on a cell array of strings.

XLS Functions Support More Formats

`xlsread` now supports Excel files having formats other than XLS (e.g., HTML) as long as the COM server is available. Also, `xlsfinfo` now returns this file format information.

Archiving Functions Accept Files on Path and ~/

Files specified as arguments to `gzip`, `gunzip`, `tar`, and `zip` can now be specified as partial path names. On UNIX machines, directories can start with `~/` or *`~username/`*, which expands to the current user's home directory or the specified user's home directory, respectively. The wildcard character * can be used when specifying files or directories, except when relying on the MATLAB path to resolve a filename or partial pathname.

sendmail No Longer Requires ASCII Messages

E-mail messages that you send using `sendmail` are no longer restricted to ASCII character encoding schemes.

MATLAB Warns on Invalid Input to str2func

Due to a bug introduced in MATLAB R14, the `str2func` function failed to issue a warning or error when called with an invalid function name or a function name that includes a path specification. In

the R2006a release, `str2func` now generates a warning under these conditions. In a future version of MATLAB, `str2func` will generate an error under these conditions.

Compatibility Considerations

Any existing code that calls `str2func` with an invalid function name or a function name that includes the path now generates a warning message from MATLAB. In a future version, this will cause an error. You should note any such warnings when using R2006a, and fix the input strings to `str2func` so that they specify a valid function name.

Changes to Character Encoding in File I/O

The `fopen` function has a new optional argument, a string that specifies a name or alias for the character encoding scheme associated with the file. If this argument is omitted or is the empty string (''), the MATLAB default encoding scheme is used. Given a file identifier as the only argument, `fopen` now returns an additional output value, a string that identifies the character encoding scheme associated with the file.

Low-level file I/O functions that read data from files, including `fread`, `fscanf`, `fgetl`, and `fgets`, read characters using the encoding scheme associated with the file during the call to `fopen`. Low-level file I/O functions that write data, including `fwrite` and `fprintf`, write characters using the encoding scheme associated with the file during the call to `fopen`.

Support for character encoding schemes has these limitations:

- Surrogate pairs are not supported. Each surrogate pair is read as a replacement character, the equivalent of `char(26)`.
- Stateful character encoding schemes are not supported.
- Byte order marks are not interpreted in any special way. Your code must skip them if necessary.
- Scanning numbers, using `fscanf`, is supported only for character encoding schemes that are supersets of ASCII. (Most popular character encoding schemes, with the exception of UTF-16, are such supersets.)

Compatibility Considerations

In V7.1 (R14SP3), low-level file I/O functions that read and write data treated characters as unsigned bytes. Programs using such functions as `fread` may have called `native2unicode` to convert input to MATLAB characters using a particular encoding scheme. Programs using such functions as `fwrite` may have called `unicode2native` to convert output from MATLAB characters using a particular encoding scheme.

For example, on a Japanese Windows platform, where the default character encoding scheme is Shift-JIS, a program may have used `native2unicode` and `unicode2native` to read and write Japanese text in this way:

```
fid = fopen(file);
data = fread(fid, '*char')';
fclose(fid);
dataU = native2unicode(data);
% operate on data
outData = unicode2native(dataU);
fid = fopen(file, 'w');
```

```
fwrite(fid, outData, 'char');
fclose(fid);
```

Such a program would produce different and possibly incorrect results in V7.2 (R2006a). The calls to `native2unicode` and `unicode2native` are no longer necessary, because the `fread` and `fwrite` functions now convert data to and from MATLAB characters using the character encoding scheme specified in the calls to `fopen`. In V7.2 (R2006a), the example code can be simplified to produce correct results:

```
fid = fopen(file);
dataU = fread(fid, '*char')';
fclose(fid);
% operate on data
fid = fopen(file, 'w');
fwrite(fid, dataU, 'char');
fclose(fid);
```

Changes to code using fread, fgets, fgetl, and fscanf

If your code calls `native2unicode` to convert input to MATLAB characters using a specified (or default) encoding scheme, you can, but do not have to, remove the calls to `native2unicode`.

This applies to reading from an encoded file using any of the following:

- `fread` with `precision` set to `'*char'` or `'char=>char'`
- `fgets` or `fgetl`
- `fscanf` with the `format` specifier set to either `'%s'` or `'%c'`

When you remove a call to `native2unicode`, be sure that the call to `fopen` supplies the same encoding argument (if any) as the call to `native2unicode`.

You may have used code like these examples in V7.1 (R14SP3):

```
indata = native2unicode(fread(fid, '*char')');
indata = native2unicode(fread(fid, 'char=>char')');
indata = native2unicode(fgets(fid));
indata = native2unicode(fgetl(fid));
indata = native2unicode(fscanf(fid, '%s'));
indata = native2unicode(fscanf(fid, '%c'));
```

You can, but do not have to, remove the calls to `native2unicode` in V7.2 (R2006a):

```
indata = fread(fid, '*char')';
indata = fread(fid, 'char=>char')';
indata = fgets(fid);
indata = fgetl(fid);
indata = fscanf(fid, '%s');
indata = fscanf(fid, '%c');
```

Changes to code using fwrite

If your code calls `unicode2native` to convert output from MATLAB characters using a specified (or default) encoding scheme, in most cases you must remove the calls to `unicode2native`. This is especially important if the encoding represents multibyte characters.

This applies to writing an encoded file using `fwrite` with `precision` set to either `'char'` or `'char*1'`.

When you remove a call to `unicode2native`, be sure that the call to `fopen` supplies the same encoding argument (if any) as the call to `unicode2native`.

You may have used code like these examples in V7.1 (R14SP3):

```
fwrite(fid, unicode2native(outbuff), 'char');
fwrite(fid, unicode2native(outbuff), 'char*1');
```

You must remove the calls to `unicode2native` in V7.2 (R2006a):

```
fwrite(fid, outbuff, 'char');
fwrite(fid, outbuff, 'char*1');
```

Graphics and 3-D Visualization

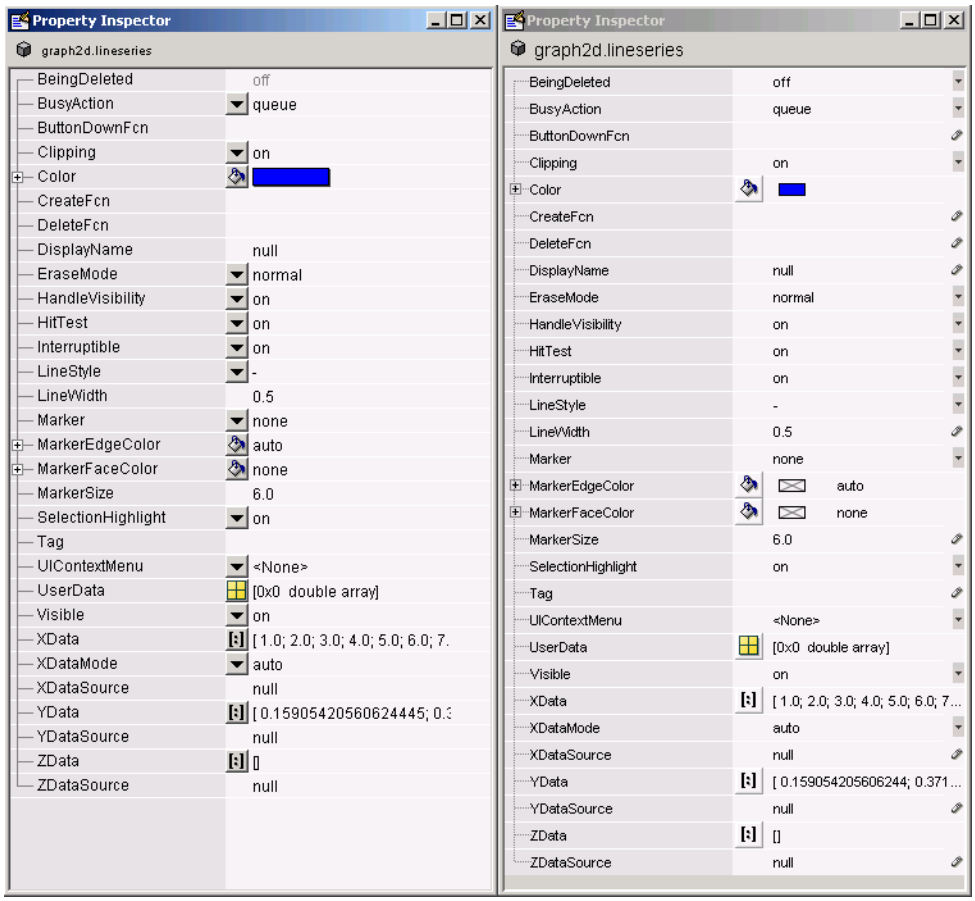
Pasting Cut or Copied Graphic Objects Can Create an Axes

The way in which MATLAB handles copying (or cutting) and pasting children of axes such as lineseries, barseries, or contourgroup objects has changed slightly. In previous releases, if no destination axes was selected prior to pasting one or more such objects, they would be pasted into the current axes (returned by the `gca` function). MATLAB no longer makes this assumption; if no axes is currently selected when you paste graphic objects, a new axes is created in the destination figure to contain them.

To avoid creating a new axes where you do not want to do so, you should be in plot edit mode and have selected a destination axes before using **Edit -> Paste** or typing **CTRL-V**. You can use the Plot Browser to conveniently select objects to copy and to paste into.

Inspector Has New Look

The Property Inspector (the GUI summoned by the MATLAB `inspect` command) has a new look, but no changed functionality. The inspector enables you to view and change the most commonly used object properties. The figure below compares the previous version (7.1, left) of the Property Inspector with the new version (7.2, right):



Note that in addition to having a smaller font and wider line spacing, the new inspector locates pop-up menus at the right margin instead of between the two columns. Also, some icons have been redesigned.

Creating Graphical User Interfaces (GUIs)

Treatment of & in Menu Label Is Changed

The use of `'&'` (ampersand) in the uimenu `'Label'` property string is changed for cases that use the constructs `'A& B'` and `'A&&B'`. The changes bring these constructs in line with the way `'&'` is used in other `'Label'` constructs. See “Compatibility Considerations” below for specific information.

Compatibility Considerations

Interpretation of `'Label'` property strings that use the following constructs is changed:

- The string `'A& B'` now produces the menu label **A& B** with no underlined mnemonic. Previously, `'A& B'` produced the label **A_B**, in which the space is a mnemonic.
- The string `'A&&B'` now produces the menu label **A & B** with no underlined mnemonic. Previously, `'A&&B'` produced the label **A&B** with no mnemonic.

If you use either construct, `'A& B'` or `'A&&B'`, in your menu labels, verify that the new resulting label is acceptable or change the `'Label'` property to a new string.

Major Documentation Revision

The MATLAB document Creating Graphical User Interfaces is reorganized and rewritten. It now consists of three sections:

- Getting Started—Leads you through the steps needed to create a simple GUI, both programmatically and using GUIDE.
- Creating GUIs with GUIDE—Contains the information, previously included in Creating Graphical User Interfaces, that you need to create a GUI using GUIDE. This section is organized in workflow order with many small examples of the various steps. A final chapter provides advanced examples.
- Creating GUIs Programmatically—For now, this section contains a summary of the available functions and complete code examples for three GUIs.

  One GUI uses a variety of user interface controls to enable a user to calculate the mass of an object after specifying the object's density and volume.

  Two other GUIs work together as an icon editor. One GUI, a color palette, is embedded in the other GUI, an icon editor. The color palette passes data to the icon editor whenever the GUI user selects a new color.

External Interfaces/API

MEX-Files Built with gcc on Linux Must Be Rebuilt

In MATLAB V7.2 (R2006a) on Linux and Linux x86-64 platforms, MEX-files built with `gcc` must be recompiled and relinked using `gcc` version 3.4 or later. Rebuilding is required because MATLAB V7.2 (R2006a) on Linux and Linux x86-64 platforms is built with `gcc` version 3.4.

Compatibility Considerations

Changes in `gcc` version 3.4 have caused incompatibilities between MATLAB V7.2 (R2006a) and MEX-files built with `gcc` versions earlier than 3.4.

On Linux and Linux x86-64 platforms, MEX-files built with `gcc` versions earlier than 3.4 cannot be used in MATLAB V7.2 (R2006a).

On Linux and Linux x86-64 platforms, MEX-files built with `gcc` version 3.4 or later cannot be used in versions of MATLAB earlier than V7.2 (R2006a).

MEX-Files in MATLAB for Microsoft Windows x64

With the introduction of MATLAB for Windows x64, you can now build 64-bit MEX-files. These MEX-files have the extension `.mexw64`. The `mexext` command returns `mexw64` in MATLAB for Windows x64.

Compatibility Considerations

MEX-files built using MATLAB for Windows (32-bit), which have `.mexw32` extensions by default, cannot be used in MATLAB for Windows x64.

By default, when MATLAB for Windows x64 is installed, the `mex.pl` and `mex.bat` scripts build MEX-files for a Windows x64 platform (with `.mexw64` extensions).

New Microsoft and Intel Compilers Supported

MATLAB V7.2 (R2006a) supports new compilers for building MEX-files on Windows and Windows x64 platforms:

- Microsoft Visual C++ 2005, also informally called Visual C++ 8.0, part of Microsoft Visual Studio 2005
- Intel Visual Fortran 9.0

Environment Variables Needed for Intel Visual Fortran

When you build a MEX-file or an Engine or MAT application using Intel Visual Fortran 9.0, MATLAB requires an environment variable to be defined, depending on whether you are building in MATLAB for Windows (32-bit) or MATLAB for Windows x64:

- MATLAB for Windows (32-bit): The environment variable `VS71COMNTOOLS` must be defined. The value of this environment variable is the path to the `Common7\Tools` directory of the Visual

Studio .NET 2002 or 2003 installation directory. (Intel Visual Fortran requires Visual Studio .NET 2002 or 2003 on 32-bit Windows platforms.) This environment variable is commonly defined by the Visual Studio .NET 2003 installation program.

- MATLAB for Windows x64: The environment variable `MSSdk` must be defined. The value of this environment variable is the path to the installation directory for Microsoft Platform SDK for Windows Server 2003. (Intel Visual Fortran requires Microsoft Platform SDK for Windows Server 2003 on Windows x64 platforms.) This environment variable is *not* commonly defined by the Microsoft Platform SDK installation program.

MWPOINTER Macro for Platform-Independent Fortran Code

MATLAB provides a preprocessor macro, `mwPointer` , that declares the appropriate Fortran type representing a pointer to an `mxArray` or to other data that is not of a native Fortran type, such as memory allocated by `mxMalloc`. On 32-bit platforms, the Fortran type that represents a pointer is `INTEGER*4`; on 64-bit platforms, it is `INTEGER*8`. The Fortran preprocessor translates `MWPOINTER` to the Fortran declaration that is appropriate for the platform on which you compile your file.

Compaq Visual Fortran Engine and MAT Options File Renamed

MATLAB V7.1 (R14SP3) included a Windows Engine and MAT options file named `df66engmatopts.bat`. This file contained options for Compaq Visual Fortran version 6.6 for use in building Fortran engine or MAT stand-alone programs. The file name `df66engmatopts.bat` originated with an earlier version of the Fortran compiler, named Digital Fortran.

In V7.2 (R2006a), this file has been renamed `cvf66engmatopts.bat` to match the Compaq Visual Fortran product name.

Compatibility Considerations

You may need to change any scripts that depend on the earlier name for the options file.

Options Files Removed for Unsupported Compilers

MATLAB V7.1 (R14SP3) included MEX, Engine, and MAT options files for a number of Windows C and Fortran compilers that were untested. These options files are not included in V7.2 (R2006a). The unsupported compilers, and the supported compilers that replace them, are:

Unsupported Compiler	Supported Replacement
Borland 5.0, 5.2, 5.3, 5.4	Borland 5.5, Borland 5.5 Free, Borland 5.6
Digital Visual Fortran 5.0, 6.0	Compaq Visual Fortran 6.1, Compaq Visual Fortran 6.6, Intel Visual Fortran 9.0
Microsoft Visual C++ 5.0, Visual C++ .NET 2002 (7.0)	Microsoft Visual C++ 6.0, Visual C++ .NET 2003 (7.1), Visual C++ 2005 (8.0)
Watcom 10.6, 11	Open Watcom 1.3

Compatibility Considerations

If you were using an untested compiler with a previous version of MATLAB, replace it with a supported compiler. You may need to recompile your MEX-files or applications.

Obsolete Functions No Longer Documented

In V7.1 (R14SP3), many MAT-file access, MX array manipulation, MEX-files, and MATLAB engine functions were declared obsolete in the External Interfaces Reference documentation. These functions are no longer documented in V7.2 (R2006a).

This section lists the obsolete functions removed from the documentation, along with replacement functions, if any.

Obsolete Functions: MAT-File Access

Obsolete Function	Replacement
`matDeleteArray (C and Fortran)`	`matDeleteVariable`
`matDeleteMatrix (C and Fortran)`	`matDeleteVariable`
`matGetArray (C and Fortran)`	`matGetVariable`
`matGetArrayHeader (C and Fortran)`	`matGetVariableInfo`
`matGetFull (C and Fortran)`	`matGetVariable` followed by `mxGetM`, `mxGetN`, `mxGetPr`, `mxGetPi`
`matGetMatrix (C and Fortran)`	`matGetVariable`
`matGetNextArray (C and Fortran)`	`matGetNextVariable`
`matGetNextArrayHeader (C and Fortran)`	`matGetNextVariableInfo`
`matGetNextMatrix (C and Fortran)`	`matGetNextVariable`
`matGetString (C and Fortran)`	`matGetVariable` followed by `mxGetString`
`matPutArray (C and Fortran)`	`matPutVariable`
`matPutArrayAsGlobal (C and Fortran)`	`matPutVariableAsGlobal`
`matPutFull (C and Fortran)`	`mxCreateDoubleMatrix` followed by `mxSetPr`, `mxSetPi`, `matPutVariable`
`matPutMatrix (C and Fortran)`	`matPutVariable`
`matPutString (C and Fortran)`	`mxCreateString` followed by `matPutVariable`

Obsolete Functions: MX Array Manipulation

Obsolete Function	Replacement
`mxClearLogical (C and Fortran)`	None
`mxCreateFull (C and Fortran)`	`mxCreateDoubleMatrix`
`mxCreateScalarDouble (C and Fortran)`	`mxCreateDoubleScalar`

Obsolete Function	Replacement
mxFreeMatrix (C and Fortran)	mxDestroyArray
mxGetName (C and Fortran)	None
mxIsFull (C and Fortran)	mxIsSparse
mxIsString (C and Fortran)	mxIsChar
mxSetLogical (C and Fortran)	None
mxSetName (C and Fortran)	None

Obsolete Functions: MEX-Files

Obsolete Function	Replacement
mexAddFlops (C)	None
mexGetArray (C and Fortran)	mexGetVariable
mexGetArrayPtr (C and Fortran)	mexGetVariablePtr
mexGetEps (C and Fortran)	mxGetEps
mexGetFull (C and Fortran)	mexGetVariable followed by mxGetM, mxGetN, mxGetPr, mxGetPi
mexGetGlobal (C and Fortran)	mexGetVariablePtr
mexGetInf (C and Fortran)	mxGetInf
mexGetMatrix (C and Fortran)	mexGetVariable
mexGetMatrixPtr (C and Fortran)	mexGetVariablePtr
mexGetNaN (C and Fortran)	mxGetNaN
mexIsFinite (C and Fortran)	mxIsFinite
mexIsInf (C and Fortran)	mxIsInf
mexIsNaN (C and Fortran)	mxIsNaN
mexPutArray (C and Fortran)	mexPutVariable
mexPutFull (C and Fortran)	mxCreateDoubleMatrix followed by mxSetPr, mxSetPi, mexPutVariable
mexPutMatrix (C and Fortran)	mexPutVariable

Obsolete Functions: MATLAB Engine

Obsolete Function	Replacement
engGetArray (C and Fortran)	engGetVariable
engGetFull (C and Fortran)	engGetVariable followed by mxGetM, mxGetN, mxGetPr, mxGetPi
engGetMatrix (C and Fortran)	engGetVariable
engPutArray (C and Fortran)	engPutVariable

Obsolete Function	Replacement
`engPutFull (C and Fortran)`	`mxCreateDoubleMatrix` followed by `mxSetPr`, `mxSetPi`, `engPutVariable`
`engPutMatrix (C and Fortran)`	`engPutVariable`
`engSetEvalCallback (C)`	None
`engSetEvalTimeout (C)`	None
`engWinInit (C)`	None

Compatibility Considerations

Most of the functions listed as obsolete in this section are unsupported in V6.5 (R13) and later versions. Some obsolete functions are unsupported in earlier versions.

If this section lists a replacement for an obsolete function, change any code that refers to the obsolete function to use the replacement instead.

If you must use an obsolete function in a MEX-file or application, use the `-V5` option to `mex` when you build the file.

Support for Licensed ActiveX Controls

MATLAB supports the use of Microsoft ActiveX controls that require licensing at both design time and runtime.

See the `actxcontrol` function for information on how to specify a design-time license key.

See Deploying ActiveX Controls Requiring Run-Time Licenses for information on how to use ActiveX Controls that require runtime licenses in your MATLAB application.

Support for VT_Date Type

MATLAB defines a data type to be used with controls requiring input defined as type VT_DATE. See Using Date Data Type for more information.

Dynamic Linking of External Libraries

MATLAB supports dynamic linking of external libraries only on 32-bit Windows systems and 32-bit Linux systems. See Calling C Shared Library Functions from MATLAB for more information.

R14SP3

Version: 7.1

New Features

Compatibility Considerations

Desktop Tools and Development Environment

Startup and Shutdown

New features and changes introduced in this version are described here.

Startup Option, –nodesktop, on Windows Platforms No Longer has Menu Bar and Toolbar; Use Function Equivalents Instead

The behavior of MATLAB software when started on Microsoft Windows platforms with the `-nodesktop` option has changed. The MATLAB Command Window no longer displays a menu bar or toolbar. This change resolves a number of problems that occurred in previous versions when running MATLAB in `-nodesktop` mode on Windows platforms.

Compatibility Considerations

Use equivalent functions instead of the menu and toolbar.

Instead of using the **File > Preferences** menu to modify the font or colors used in the Command Window, run `preferences -nodesktop`. For more information, see “preferences Function Now Supports -nodesktop Option” on page 34-4.

Desktop

New features and changes introduced in this version are organized by these topics:

Arranging Windows and Documents

Figure Windows Now Dockable on Macintosh Platforms

On Apple Macintosh platforms, figure windows are now dockable.

Resize Multiple Tools at Once

You can now position the pointer at the intersection of three or four tools or documents to resize all of them at once.

Resize and Move Desktop Tools Using the Keyboard

There are now menu items you can select to move and resize the active tool in the desktop. Use the menu item mnemonics to perform those action with the keyboard. For example, if the Command Window is in the desktop along with other tools, press **Ctrl+0** (or click in it) to make the Command Window the active tool. Then press **Alt+D**, **V**, which is the mnemonic equivalent for selecting **Desktop > Move Command Window**. The pointer becomes an arrow. Use the arrow keys to move an outline of the Command Window to a new dockable location. Press **Enter** to dock it there, or press **Esc** to return the Command Window to its original position.

Resize Names in the Document Bar

You can now adjust the width of a name in the document bar when the bar is at the top or bottom of the window.

Positioning Document Bar Menu Item Name Changed

In previous versions, selecting **Desktop > Document Bar** displayed only menu items for positioning the document bar. Now, there are additional menu items. The same change was made to the context menu for the document bar. To access the menu items for positioning the document bar, select **Desktop > Document Bar > Bar Position**.

Keyboard Access Added for Document Bar Options

The **Desktop > Document Bar** now includes these items: **Alphabetize**, **Width**, and **Move documentname On Bar**. With their inclusion in the menu, you can use the keyboard to access these features via mnemonics. For example, on Windows platforms, press **Alt+D**, **M**, **A** as a shortcut to for **Desktop > Document Bar > Alphabetize**.

Left/Right and Top/Bottom Split for Document Arrangements Name Changed

When arranging documents in desktop tools, you can choose **Window > Left/Right Split** or **Window > Top/Bottom Split** to show two documents at once in the tool. Those menu items are now called **Left/Right Tile** and **Top/Bottom Tile**. This change was made to avoid any confusion with the Editor/Debugger's new split screen feature.

Preferences Directory Added for R14SP3; Supplements R14 Directory

There is a new preferences directory, `R14SP3`. This is the directory name returned when you run the `prefdir` function. When you install R14SP3, MATLAB migrates files from your existing preference directory, `R14`, to the new directory, `R14SP3`. Changes made to files in the directory when you run R14SP3 are not used when you run previous R14 releases.

This represents a change in the preference directory MATLAB uses for a minor release, and was done to prevent serious backwards compatibility problems. It is primarily relevant if you use R14SP3 and previous R14 releases. If you only run R14SP3, or run R14SP3 with R13 or R12 releases, you will not be affected by this change.

In the past, minor releases and the associated major release used the same preferences directory. For example, R13 and R13SP1 shared the `R13` preferences directory. That continues to be true for all previous releases, but is not true for R14SP3 and beyond. The `R14` preferences directory will be shared by the R14 through R14SP2 releases, but the new `R14SP3` preferences directory is only used by R14SP3. This means that changes made to files in the directory while running R14SP3 re not used when you run a previous R14 releases, and the reverse is true. For example, statements added to the Command History when you run R14SP3 are not in the Command History when you run R14SP2.

For more information, see the reference page for `prefdir`.

Compatibility Considerations

This change was made to prevent major backwards compatibility problems. Use the `R14SP3` preferences directory instead of the `R14` directory. If you use the `prefdir` function and have code that relies on the result being `R14`, you will need to modify that code.

Preferences Changes for Fonts, Hyperlinks, and -nodesktop

Font Antialiasing Preference Added

In **Preferences** > **Fonts**, select the new antialiasing preference to provide a smoother appearance to desktop fonts.

Hyperlink Color Preference Changed

There is a new **Colors** preference for specifying the color of hyperlinks in the Command Window and the Help browser **Index** pane. In previous releases, this preference only applied to the Command Window hyperlinks and was accessed via Command Window preferences.

preferences Function Now Supports -nodesktop Option

Run `preferences -nodesktop` after starting MATLAB on Windows platforms with the `-nodesktop` option to change Command Window font and colors via a special **Preferences** dialog box.

To set other available preferences for the Command Window after starting MATLAB with the `-nodesktop` option, run `preferences` and use the resulting **Preferences** dialog box for all tools and products. Note that changes you make to font and color preferences in this dialog box do not apply to the Command Window.

info.xml File Automatic Validation; Shows Warnings for Invalid Constructs

If you add your own toolbox to the **Start** button, you can use the schema file for its `info.xml` file, *`matlabroot`*`/sys/namespace/info/v1/info.xsd`. MATLAB now automatically validates your `info.xml` file against this schema when you click the **Start** button after updating and refreshing your `info.xml` file.

Compatibility Considerations

If your `info.xml` contains invalid constructs, you will see warnings in the Command Window until you correct the problems.

Other Desktop Changes

Paste Special Menu Item Renamed

In the **Edit** menu, the name of the **Paste Special** item has been replaced by **Paste to Workspace**, but the functionality remains the same. It opens the Import Wizard so you can paste the clipboard contents to the workspace in MATLAB.

Rename Shortcut Categories

You can now rename shortcut categories.

Running Functions — Command Window and Command History

New features and changes introduced in this version are

- “Tab Completion Preference Added” on page 34-5
- “Tab Completion No Longer Shows Entries Twice” on page 34-5

Tab Completion Preference Added

There is a new Command Window preference, **Tab key narrows completion**. When selected, with a list of possible completions in view, type another character and press **Tab** to further narrow the list shown. Repeat to continue narrowing the list. This behavior is similar to tab completion behavior in releases prior to R14.

Tab Completion No Longer Shows Entries Twice

In previous versions, when completing file names or function names, a name sometimes appeared twice in the completion list, once with the file extension and once without. Now the entry appears only once.

Incremental Search Now Supports Removing Characters

In incremental search, use **Ctrl+G** to remove characters back to the previous successful string of characters found. For example, when searching for the term `plode`, the text is not found and `Failing` appears in the incremental search field. **Ctrl+G** automatically removes the `de` from the search term because `plo` does exist in the file.

Hyperlink Color Preference Moved

The preference for specifying the hyperlink color has moved from the Command Window preferences pane to the **Colors** preferences pane. The hyperlink color now also applies to links in the Help browser **Index** pane.

Compatibility Considerations

Use the **Colors** preference pane to specify the hyperlink color, and be aware that it also impacts the Help browser Index pane color.

Help

New features and changes introduced in this version are

Hyperlink Color in the Index Pane Preference Added

You can now specify the color for links in the Help browser **Index** pane using the **Colors** preferences pane. The hyperlink color also applies to links in the Command Window, so changes you make to the preference apply to both tools.

New Look for Demos, Including Thumbnails and Categories

Stylistic changes were made to the Demos interface in the Help browser. On the summary page for a product, each demo appears with a thumbnail image that provides an indication of the type of output it creates, as well as an icon representing the type of demo (M-file, M-GUI, model, or video).

Demos Run in Command Window as Scripts and Their Variables Now Created in Base Workspace

In this release, all M-file demos include the **Run in the Command Window** link, which executes the demo via `echodemo`.

In previous releases, some M-file demos provided a **Run** hyperlink in the display pane. When you clicked **Run**, the M-file demo executed in a GUI via the `playshow` function. An example of this type of demo is the MATLAB Mathematics Basic Matrix Operations demo, `intro.m`. In this release, the **Run** hyperlink for these M-file demos has been replaced by **Run in the Command Window**. It executes the demo step by step in the Command Window via the `echodemo` function. Double-clicking this type of M-file demo in the Navigator pane no longer runs the M-file demo, but opens the M-file in the Editor/Debugger where you can run it step by step using **Cell > Evaluate Current Cell and Advance**.

Compatibility Considerations

The new **Run in Command Window** hyperlink represent a change in the way demos run.

The `echodemo` function MATLAB uses to run M-file demos in the Command Window runs the demos as scripts. The `playshow` function MATLAB used to run M-file demos in previous releases ran the demos as a function. This means that now the demo's variables are created in the base workspace. If you have variables in the base workspace when you run an M-file demo, and the demo uses an identical variable name, there could problems with variable name conflicts. For example, your variable could be overwritten. The demo's variables remain in the base workspace after the demo finishes running until you clear them or quit MATLAB. Another change is that figures are not automatically closed when you end the demo.

echodemo Function Added to Replace playshow function

There is a new `echodemo` function that replaces `playshow`. The Demos browser uses `echodemo` to execute M-file demos when you click the **Run in the Command Window** link.

Compatibility Considerations

The `playshow` function is deprecated in favor of the `echodemo` function. In a future release, the `playshow` function will be removed. In practice, both `echodemo` and `playshow` are helper functions for running demos. It is unlikely you would ever call either `playshow` or `echodemo` directly, and especially not in M-files.

Add Demos to Favorites

You now can add published M-file demos to favorites.

Adding Your Own Demos Type Tag Now Supported

If you add demos for your own toolbox, you can use the new `<type>` tag for a `<demoitem>` to identify the type of demo in your toolbox's `demos.xml` file.

Bug Reporting System Introduced

You now can view bugs fixed with this release, as well as any known bugs using the Bug Reports database in the Support section of the MathWorks Web site. The MathWorks continuously updates the database to add any newly found bugs and compatibility issues, as well as any new workarounds and solutions. The system includes bugs found and fixed in R14SP2 and later releases.

Workspace, Search Path, and File Operations

New features and changes introduced in this version are described here.

Find Files Offers Additional Filtering

The Find Files tool has been enhanced. It now allows you to search all file types except those specified. It also lets you ignore files larger than a specified size. Along with enhancements to the Find Files tool, some minor feature changes were made, including the removal of the **Restore Defaults** button.

Visual Directory View to be Removed

In the next release, the Current Directory browser will no longer support the Visual Directory view (accessed using the  toolbar button).

Compatibility Considerations

Some features currently available using the Visual Directory view will not be available in the next release when the feature is removed.

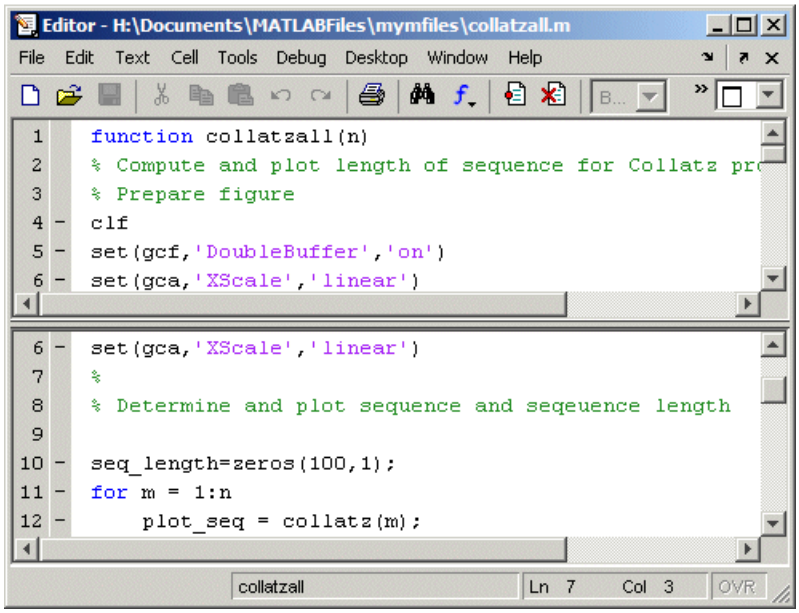
Editing and Debugging M-Files

New features and changes introduced in this version are

Split Screen Display Added

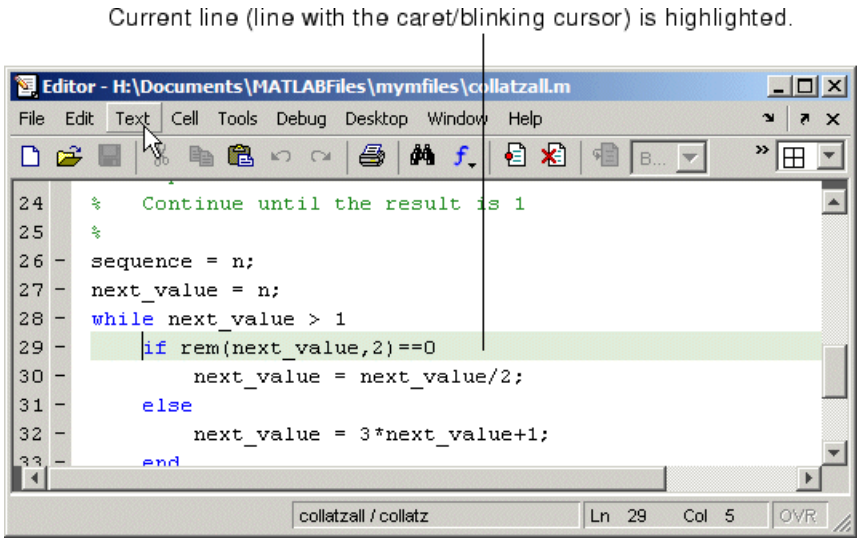
The Editor/Debugger now supports a horizontal or vertical split screen for displaying two different parts of the same document at once. To split the screen, select **Window > Split Screen** and the splitting action you want, for example, **Top/Bottom**. Alternatively, drag the splitter bar that appears above the vertical scroll bar or to the left of the horizontal scroll bar. To remove the splitter, drag it to an edge of the window.

Document with top/bottom split.



Highlight Current Line Added

You can set a preference to highlight the current line, that is, the line with the caret (also called the blinking cursor). This is useful, for example, to help you see where copied text will be inserted when you paste. To highlight the current line, select **Preferences > Editor/Debugger > Display** and under **General Display Options**, select the check box for **Show caret row highlighting**. You can also specify the color used to highlight the line.



Comment Lines in Java, C, or C++ Program Files Now Supported

You can now use the **Text** > **Comment** feature to comment selected lines in Sun Microsystems Java, ANSI® C, and C++ program files. This adds the `//` symbols at the start of the selected lines. Similarly, **Text** > **Uncomment** removes the `//` symbols from the front of selected lines in Java, C, and C++ program files.

HTML File Indenting Feature Added as the Default

There is a new Editor/Debugger language preference for HTML files to specify block indenting. By default, the preference is selected so block indenting applies when typing text in HTML files.

In addition, you now can select **Text** > **Smart Indent** to apply smart indenting to selected text in HTML files.

Compatibility Considerations

When typing text in HTML files, you will automatically see block indenting because the preference is selected by default.

Incremental Search Now Supports Removing Characters

In incremental search, use **Ctrl+G** to remove characters back to the previous successful string of characters found. For example, when searching for the term `plode`, the text is not found and `Failing` appears in the incremental search field. **Ctrl+G** automatically removes the `de` from the search term because `plo` does exist in the file.

Emacs Key Binding for Select All

With the Emacs key bindings preference selected, use **Ctrl+X**, **H** to select all.

Change Case Added to Menu

Use new items in the **Text** menu to change the case of selected text. You can also use the keyboard equivalents for changing case that existed in previous versions—these are shown in the menu next to each item.

Nested Function Name No Longer in Status Bar

The Editor/Debugger no longer displays the current nested function name in the status bar. Look in the M-file to view the current nested function name.

Tuning and Managing M-Files

New features and changes introduced in this version are described here.

Directory Reports Uses New Run Buttons

With Directory Reports displayed in the Web browser, you can use these two new buttons:

- **Rerun This Report** — This updates the currently displayed report after you have made changes to the report options or to any files in the current directory.
- **Run Report on Current Directory** — Use this after changing the current directory to run the same type of report for the new current directory.

These new buttons replace the **Refresh** button.

Override %#ok with the New mlint -notok Option

There is a new option for the `mlint` function, `'-notok'` you can use to override any statements that include `%#ok` (the symbol you add to the end of a line instructing `mlint` to ignore the line). That is, `mlint` will run for all lines in the file and will not ignore any statements.

Hyperlink Now Part of Messages Displayed by mlint

When you run the `mlint` function, the line number in the messages displayed is a hyperlink that when clicked, opens the file in the Editor/Debugger scrolled to that line number.

Profiler Button Added to Toolbar

There is now a button  on the MATLAB desktop toolbar to open the Profiler.

Publishing Results

New features and changes introduced in this version are described here.

Notebook Setup Changes; Some Arguments Removed

The `notebook` function setup behavior and syntax have changed.

When you run `notebook('-setup')`, MATLAB automatically obtains all the information about your Microsoft Word application from the system registry for yourWindows environment and you are no longer prompted to supply the information.

In previous versions, when you configured Notebook, you ran

```
notebook ('-setup')
```

Notebook then prompted you to specify the version of Word you were using, and if needed, the location of Word and its template directory. You could supply the information using optional arguments to the `notebook` function:

```
notebook('-setup', wordversion, wordlocation, templatelocation)
```

Now, when you run `notebook('-setup')`, MATLAB automatically obtains all the Word information from the registry for yourWindows environment.

Compatibility Considerations

If you use `notebook` with the `wordversion`, `wordlocation`, and `templatelocation` arguments in any of your files (for example, `startup.m`), remove those arguments in your files. If you specify the optional arguments, the `notebook` function runs and issues a warning, but ignores the values. In a future release, MATLAB will issue an error when it encounters `notebook` with these arguments.

Versions of Microsoft Word Application Supported by Notebook; Microsoft Word 97 No Longer Supported

MATLAB Notebook supports the Microsoft Word version 2000 application. Notebook also supports the Microsoft Word 2002 application and Microsoft Word 2003 application, both for the Microsoft Windows XP platform.

Compatibility Considerations

As of MATLAB 7.1 (R14SP3), Notebook no longer supports the Microsoft Word 97 application.

Mathematics

New Functions

The following functions are new in R14SP3:

Function	Description
`hypot`	Square root of sum of squares
`mode`	Finds most frequent values in sample

Compatibility Considerations

A new function name can potentially introduce a backward incompatibility since it can, under certain circumstances, override a variable with the same name as the new function. This is especially true for names that are commonly used as variable names in program code.

An example of such a function name is the `mode` function, introduced in this release. If you have M-file programs that use `mode` as a variable name, it is possible under certain conditions for MATLAB to interpret these variable names as function names by mistake. Read the section Variable Names in the MATLAB Programming documentation to find out how to avoid having these variables misinterpreted.

If your program code uses a user-written function named `mode`, you may find that MATLAB calls the new MATLAB `mode` function instead of your own `mode` function. To correct this, modify your MATLAB path by placing the location of your own `mode` function closer to the beginning of the path string than the location of the MATLAB `mode.m` file. The help for the `addpath` and `rmpath` functions explains how to modify your MATLAB path.

Modified Functions

The following functions have been modified in MATLAB 7.1:

Function	Modified Behavior
`accumarray`	Allows more flexibility for input/output classes and functions to be called
`odeset`	New `NonNegative` integration property to impose nonnegativity constraints on an ODE solution
`rand`	Supports the Mersenne Twister algorithm in generating random numbers
`svd`	Returns economy decomposition

Changes to accumarray

MATLAB Version 7.1 adds the following new features to the `accumarray` function:

- The data type for the `val` input can be any numeric type, or logical, or character.
- The data type for the `subs` input can be any numeric type.
- You can use a cell array of separate index vectors for the `subs` input.
- When you specify a `function` input argument, the value returned by `accumarray` is given the same class as the values returned by that function.

- You can control the sparsity of the value returned by `accumarray` by specifying the new input argument `issparse`.

Imposing Nonnegativity Constraints on Computed ODE Solution

There is a new integration property called `NonNegative` that you can use when applying ODE initial value problem solvers. If you need to solve a problem in which certain components of the solution must be nonnegative, use the `NonNegative` property to impose nonnegativity constraints on the computed solutions.

See Nonnegative Solutions under Calculus in the MATLAB Mathematics documentation for more information on this feature.

Mersenne Twister Support in rand

The `rand` function now supports a method of random number generation called the Mersenne Twister. The algorithm used by this method, developed by Nishimura and Matsumoto, generates double precision values in the closed interval `[2^(-53), 1-2^(-53)]`, with a period of `(2^19937-1)/2`.

For a full description of the Mersenne twister algorithm, see `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`.

svd Returns Economy Decomposition

The following feature was released in MATLAB 7.0, but was undocumented until this release.

The command `svd(A,'econ')` returns economy decomposition on matrices having few rows and many columns as well as those with many rows and few columns. `svd(A,0)` continues to behave as it always has, namely to only return economy-sized decomposition on matrices having many rows and few columns.

Note that this does not carry over to the `qr` function as there is no valid way of cutting out any of the information returned by `qr` to make an economy-sized decomposition of matrices having few rows and many columns.

New Location for LAPACK Libraries

The location of the LAPACK libraries has been changed. These libraries are now located in

```
extern/lib/win32/microsoft/libdflapack.lib
extern/lib/win32/microsoft/libmwlapack.lib
```

This change impacts you only if you build MEX-files that call LAPACK and BLAS functions.

Documentation on Data Analysis

The section of the MATLAB Mathematics documentation on “Data Analysis and Statistics” has been moved to a new MATLAB Data Analysis book. This book documents MATLAB functions and tools that support basic data analysis, including plotting, descriptive statistics, correlation, interpolation,

filtering, and Fourier analysis. It also documents the new object-oriented command-line API for analyzing time-series data.

Data Analysis

Data Analysis Documentation

The MATLAB 7.1 documentation includes a new Data Analysis book that describes how to use MATLAB functions and tools for common data-analysis tasks:

- Plotting
- Filtering
- Interpolation
- Descriptive statistics
- Correlation
- Data fitting using linear regression
- Fourier analysis
- Time-series analysis

Some of the content in Data Analysis is incorporated from the Mathematics and Graphics books, such as data plotting, descriptive statistics, data fitting, and Fourier analysis. All information about time-series analysis is new.

Time-Series Analysis

You can analyze time-series data using the new `timeseries` and `tscollection` objects and methods, as well as the Time Series Tools graphical user interface. This new functionality supports the following:

- Representation for univariate or multivariate MATLAB time series and Simulink logged-signals data
- Built-in management of time units
- Removal or interpolation of missing data
- Resampling of data
- Arithmetic operations for `timeseries` objects
- Synchronization of time series

Note Due to reported instabilities on the Linux 64 platform, you must manually enable the Time Series Tools feature before starting Time Series Tools.

To manually enable Time Series Tools on the Linux 64 platform, type the following at the MATLAB prompt:

```
rehash toolboxcache
feature('TimeSeriesTools',1)
```

Programming

New Functions

This version introduces the following new functions:

Function	Description
`arrayfun`	Applies a given function to each element of an array. This is especially useful for arrays of structures.
`exifread`	Reads EXIF information from JPEG and TIFF image files
`structfun`	Applies a given function to each field of a structure
`swapbytes`	Swaps byte ordering
`typecast`	Converts data types without changing underlying data

Compatibility Considerations

A new function name can potentially introduce a backward incompatibility since it can, under certain circumstances, override a variable with the same name as the new function. This is especially true for names that are commonly used as variable names in program code. Read the section Variable Names in the MATLAB Programming documentation to find out how to avoid having these variables misinterpreted.

Modified Functions

The following functions were modified in this version:

Function	Modified Behavior
`cellfun`	Applies a given function to each cell of a cell array
`datestr`	Seconds field truncates instead of rounding
`error`	Saves stack information that you can retrieve using `lasterror`
`isfield`	Supports cell array input
`lasterror`	Returns stack information on last error
`rethrow`	Accepts stack information as input
`who`, `whos`	Displays information separately for nested functions

Compatibility Considerations

The following functions might, under certain circumstances, return a different value than what was returned in MATLAB 7.0.4 (R14SP2):

- `datestr`: Output might differ by 1 second from what was returned in a previous version.
- `isfield`: Output might differ if you used this feature in a release in which it was not officially supported.

Evaluation Functions for Arrays, Structures, Cells

MATLAB offers the capability to apply a given function to each element of an array, each field of a structure, or each cell of a cell array. See the help on `arrayfun`, `structfun`, and `cellfun` for more information.

Using who and whos with Nested Functions

When you use `who` or `whos` inside of a nested function, MATLAB returns or displays all variables in the workspace of that function, and in the workspaces of all functions in which that function is nested. This applies whether you include calls to `who` or `whos` in your M-file code or if you call `who` or `whos` from the MATLAB debugger. See the`who` reference page for more information.

Date and Time Functions Support Milliseconds

The `datestr`, `datenum`, and `datevec` functions now support time specification in milliseconds. Use the symbol `.FFF` to represent milliseconds in any of these three functions. See the table labeled Free-Form Date Format Specifiers on the `datestr` reference page for more information.

Stack Trace Provided for lasterror

The `lasterror` function now returns an additional field in the structure that it returns. The new `stack` field contains information from the stack on the M-file, function, and line in which the error occurred.

You can use this stack information to track down the source of an error, or as an input to the `rethrow` function. When used with `rethrow`, MATLAB sets the stack of the rethrown error to the value contained in the `stack` input.

isfield Function Supports Cell Arrays; Results Might Differ from Previous Version

The `isfield` function now supports cell array input as shown in this example. Check structure `S` for any of four possible field names. In this case, only the first is found, so the first element of the return value is set to true:

```
S = struct('one', 1, 'two', 2);

fields = isfield(S, {'two', 'pi', 'One', 3.14})
fields =
     1     0     0     0
```

Compatibility Considerations

There might be backward compatibility issues associated with this change if you used `isfield` with cell array input in a previous release. In previous releases, although `isfield` might have worked with this type of input in certain cases, it was not officially a supported feature. If you used this previously unsupported syntax in previous releases, you may see a change in the content and/or size of the return values in this release.

For example, create a structure `s` with three fields `a`, `b`, and `c` created in that order. In MATLAB 7.0.4, `isfield` called with a cell array input returns `true` if any of the elements of the cell array matches a

field name, and if that element is in the same position in the cell array as the field is in the structure. This is true for `'c'`:

```
isfield(s, {'b'; 'a'; 'c'})
ans =
     1
```

In MATLAB 7.1, `isfield` returns `true` for each element in the cell array that matches a field name, regardless of where the string is positioned in the cell array. This is true for `'a'`, `'b'`, and `'c'`:

```
isfield(s, {'b'; 'a'; 'c'})
ans =
     1
     1
     1
```

Support for Reading EXIF Data from Image Files

You can now read EXIF (Exchangeable Image File Format) data from JPEG and TIFF graphics files using the new `exifread` function. EXIF is a standard used by digital camera manufacturers to store information in the image file, such as the make and model of a camera, the time the picture was taken and digitized, the resolution of the image, exposure time, and focal length.

Performance Improvements to the MATLAB JIT/Accelerator on Macintosh

The JIT/Accelerator for MATLAB, introduced in MATLAB Version 6.5 for Windows and UNIX, is now also supported on Macintosh systems. The JIT/Accelerator affects the performance of MATLAB and can give you a substantial performance increase over earlier MATLAB versions for many MATLAB applications.

Specifying fread Precision as Number of Bits

The following information on the `fread` function applies to MATLAB 7.1 and also to earlier versions.

MATLAB provides the following method of specifying a `precision` argument in a call to `fread`:

```
input_format=>output_format
```

For example, to read 50 8-bit unsigned integers from a file and convert them to characters, you can use

```
c = fread(fid, 50, 'uint8=>char')'
```

If the input format and output format are the same, you can abbreviate the precision specifier by using

```
*input_format
```

For example, you can replace

```
c = fread(fid, 50, 'uint8=>uint8')'
```

with

```
c = fread(fid, 50, '*uint8')'
```

You can also use this notation with an input stream that is specified as a number of bits (e.g., `bit4` or `ubit18`). MATLAB translates this into an output type that is a signed or unsigned integer (depending on the input type), and which is large enough to hold all of the bits in the source format. For example, `*ubit18` does not translate to `ubit18=>ubit18`, but instead to `ubit18=>uint32`.

Seconds Field Now Truncated; Results Might Differ

When handling time data, MATLAB now truncates the `seconds` field instead of rounding it. This is consistent with the way that MATLAB handles `hours` and `minutes`.

For example, using MATLAB 7.0.4 (R14SP2), `datestr` returns

```
t = datestr('11:30:01.666')
t =
   01-Jan-2005 11:30:02
```

while MATLAB 7.1 (R14SP3) returns

```
t = datestr('11:30:01.666')
t =
   01-Jan-2005 11:30:01
```

Compatibility Considerations

If your M-files relied on the previous behavior, you might get different results.

Built-in Functions No Longer Use .bi; Impacts Output of which Function

In previous releases, MATLAB function dispatching located built-in functions by means of special files having a `.bi` file extension. MATLAB no longer uses this mechanism to locate built-in functions. All `.bi` files have been removed in MATLAB 7.1.

Compatibility Considerations

If you have M-files that relied on built-in files having a `.bi` extension, your files need to accommodate this change.

There are changes in how MATLAB displays built-in functions using `which`:

In MATLAB 7.0.4 (R14SP2),

```
which -all int32
\\matlab\toolbox\symbolic\@sym\int32.m          % sym method
\\matlab\toolbox\matlab\datatypes\int32.bi      % Shadowed
\\matlab\toolbox\matlab\datatypes\int32.m       % Shadowed
```

In MATLAB 7.1 (R14SP3),

```
which -all int32
built-in (\\matlab\toolbox\matlab\datatypes\int32)
\\matlab\toolbox\symbolic\@sym\int32.m          % sym method
```

New Warning About Potential Naming Conflict

If you change directories (`cd`) or add a new directory to your current MATLAB path, and the new directory contains an M-file having the same name as a MATLAB built-in function, MATLAB now displays a warning alerting you to the potential naming conflict. For example,

```
Warning: Function D:\test\matlab\disp.m has the same name as a
MATLAB builtin. We suggest you rename the function to avoid a
potential name conflict.
```

In general, any file system event that leads to path refreshing in MATLAB can trigger this warning if the directory involved in this event has such a user function under it.

Compatibility Considerations

MATLAB might generate warnings about naming conflicts that did not appear in previous versions. To avoid this warning, renaming your M-files that have name conflicts with built-in functions.

Graphics and 3-D Visualization

Plot Tools Now Available on Mac Platform

As a consequence of enabling Java figures on Macintosh, the Plot Tools user interface is now available to Mac users, enabling them to interactively add data to plots, change plot symbology, and otherwise customize their data plots.

Documentation for Data Analysis Reorganized

Documentation explaining techniques for analyzing graphical data has been shifted from the Graphics book of the MATLAB documentation to a new book called Data Analysis.

Creating Graphical User Interfaces (GUIs)

Plans for Obsolete Functions

The table below indicates functions that were designated as obsolete prior to R14SP3 and that will be removed in a future version.

Compatibility Considerations

If you use these functions, you should use replacement functions instead.

Obsolete Function	Removed from Version	Replacement
`clruprop`	Future version	`rmappdata`
`ctlpanel`	Future version	`guide`
`extent`	Future version	`get(txtobj,'extent')`
`figflag`	Future version	`findobj` to determine if figure exists. `figure(fighandle)` to bring figure to front and give it focus.
`getuprop`	Future version	`getappdata`
`hthelp`	Future version	`web`
`layout`	Future version	None provided
`matq2ws`	Future version	None provided
`matqdlg`	Future version	None provided
`matqparse`	Future version	None provided
`matqueue`	Future version	None provided
`menuedit`	Future version	`guide`
`menulabel`	Future version	Use `'&'` to specify mnemonics and `'Accelerator'` property to define accelerator keys.
`setuprop`	Future version	`setappdata`
`wizard`	Future version	None provided
`ws2matq`	Future version	None provided

External Interfaces/API

mex Switches Now Supported on Microsoft Windows

MATLAB now supports the `-l` and `-L` options to the `mex` command on Windows operating systems. In previous releases of MATLAB, these options were supported only on UNIX systems.

Switch	Description
`-l`	Specifies additional libraries to link against. **Note** On Windows operating systems, the `-l` option can specify libraries of two forms. For example, specifying `-l name` matches either `name.lib` or `libname.lib`, whereas on UNIX it matches only `libname.lib`.
`-L`	Specifies a path to use when MATLAB searches for library files specified with the `-l` option. The `-L` option must precede the `-l` option.

For the switches you can use with the `mex` command, see the MEX Script Switches table in the Custom Building MEX-Files section of Creating C/C++ and Fortran Programs to be Callable from MATLAB (MEX-Files) in the MATLAB External Interfaces documentation.

New COM Programmatic Identifier

There is now a `ProgID` that enables you to use the full desktop version of MATLAB as an automation server.

`Matlab.Desktop.Application` starts an automation server using the most recent version of MATLAB that is installed on your system.

New File Extension for MEX-Files on Windows Systems

MATLAB now uses the extension `.mexw32` for MEX-files on 32-bit versions of Windows systems. In previous versions, MATLAB used the extension `.dll`.

MathWorks recommends that you recompile all MEX-files after installing MATLAB 7.1. MEX-files compiled in MATLAB 7.0.4 with `.dll` extensions should still work in MATLAB 7.1.

There may be two MEX-files with the same name, except that one has a `.mexw32` extension and the other has a `.dll` extension. When these files are both on the MATLAB search path:

- If the two files are in the same directory, MATLAB uses the `.mexw32` file.
- If the two files are in different directories, MATLAB uses the file in the directory that is higher on the search path.

If you want one of these two files to take precedence over the other, ensure that the directory that contains the file you want MATLAB to use is higher on the search path than the directory that contains the file you do not want MATLAB to use.

Compatibility Considerations

Previous versions of MATLAB do not recognize MEX-files compiled in MATLAB 7.1 with `.mexw32` extensions. However, you can use the `mex -output` option in MATLAB 7.1 to build a MEX-file with a `.dll` extension that earlier versions of MATLAB can recognize.

You may need to update any MATLAB scripts or makefiles that explicitly expect `.dll` extensions for compiled MEX-files. You can use the `mexext` function in MATLAB to obtain the extension for the platform and version you are working on. A new `mexext` script obtains the appropriate extension when executed from outside MATLAB, as in a makefile.

On Windows systems, MATLAB issues warnings at MEX setup time, compile time, and run-time to notify you of possible incompatibilities resulting from the change in MEX-file extension from `.dll` to `.mexw32`.

New mex–output Behavior for Compatibility

The `-output` option to `mex` specifies the filename of the compiled MEX-file. In general, `mex` ignores any filename extension supplied in the `-output` argument and uses the extension for the compiled file that is appropriate for the architecture. However, on Windows systems, if the `-output` argument specifies a `.dll` extension, the compiled file has this extension instead of `.mexw32`. Previous versions of MATLAB can recognize the resulting compiled file.

Conflicting MEX-Files Renamed Automatically

If two files with the same name but with `.mexw32` and `.dll` extensions exist in the same directory, MATLAB uses the `.mexw32` file. To avoid unintended shadowing, MATLAB automatically renames compiled MEX-files under the following circumstances:

- When you build a MEX-file with a `.mexw32` extension and the directory contains an existing file with the same name, but with a `.dll` extension, the extension of the `.dll` file is changed to `.dll.old`.
- When you build a MEX-file with a `.dll` extension (using the `mex -output` option) and the directory contains an existing file with the same name, but with a `.mexw32` extension, the extension of the `.mexw32` file is changed to `.mexw32.old`.

New Return Value for mexext on Windows Systems

On 32-bit Windows platforms, the `mexext` function now returns `mexw32`. In MATLAB 7.0.4 it returned `dll`.

New mexext Script to Obtain MEX-File Extension in Makefiles

A new script displays the MEX-file extension in the current version of MATLAB that corresponds to the platform on which the script is executed. It is intended to be used outside MATLAB, in makefiles or scripts, to obtain the appropriate filename extension for compiled MEX-files. Use this script instead of explicitly specifying the MEX-file extension in a makefile.

The script is named `mexext.bat` on Windows platforms and `mexext.sh` on UNIX platforms. It is located in the directory `$matlab/bin`, where `$matlab` represents the string returned from the `matlabroot` command.

The script displays the MEX-file extension without a leading period. For example, on 32-bit Windows platforms, it returns `mexw32`.

Following is a fragment of a GNU makefile that uses the `mexext` script to obtain the MEX-file extension:

```
ext = $(shell mexext)

yprime.$(ext) : yprime.c
    mex yprime.c
```

New Preferences Directory and MEX Options

The MATLAB preferences directory has changed. In MATLAB 7.1, the preferences directory is named `R14SP3`. In previous R14 releases, the preferences directory was named `R14`. For more information, see the documentation for `prefdir`, which returns the preferences directory.

Compatibility Considerations

When you install MATLAB 7.1, MATLAB migrates some files from any existing `R14` preferences directory to the new `R14SP3` directory. However, MATLAB does not migrate the MEX options file, `mexopts.bat`. If you want to preserve any MEX options that you have customized in an earlier R14 release, you need to migrate your options to the new `R14SP3` preferences directory.

You can migrate your MEX options in either of two ways:

- If you have customized only a few options: Invoke `mex` with the `-setup` option to create a new `mexopts.bat` file in the `R14SP3` preferences directory. Edit the new `mexopts.bat` file to customize the MEX options there.
- If you have customized many options: Copy your customized `mexopts.bat` file from the old `R14` preferences directory to the new `R14SP3` directory. Edit at least the settings of the LIBLOC and NAME_OUTPUT linker parameters in the `mexopts.bat` file. These lines should look as follows on Windows systems when using a Microsoft compiler:

  ```
  set LIBLOC=%MATLAB%\extern\lib\win32\microsoft
  set NAME_OUTPUT=/out:"%OUTDIR%%MEX_NAME%%MEX_EXT%"
  ```

The `LIBLOC` parameter has changed because import libraries have moved; see “Import Libraries Moved” on page 34-26. The value of this parameter depends on the platform you are running MATLAB on and the vendor of the compiler you are using.

The `NAME_OUTPUT` parameter has changed because the extension for compiled MEX-Files has changed on Windows systems; see “New File Extension for MEX-Files on Windows Systems” on page 34-23.

Compiler Support

The set of compilers that MATLAB supports has changed in MATLAB 7.1. For an up-to-date list of supported compilers, see the Supported and Compatible Compilers Web page.

Compatibility Considerations

You may need to recompile code compiled with an earlier compiler that is no longer supported.

Import Libraries Moved

The import libraries (`.lib` files) for the MATLAB `dll` files have been moved up a directory level and are no longer specific to the compiler version. The new location for these files is

```
$matlab/extern/lib/$arch/$vendor
```

where the terms `$matlab`, `$arch`, and `$vendor` respectively represent the string returned from the `matlabroot` command, the platform you are running MATLAB on, and the vendor of the compiler you are using.

Compatibility Considerations

You may need to change any code that depends on the previous library locations.

MEX Perl Script Moved

The MEX Perl script used in building MEX-files is now located in `$matlab/bin`, rather than `$matlab/bin/win32`. (The term `$matlab` represents the string returned by the `matlabroot` function.) You should not notice any difference, however, as a batch file located in `$matlab/bin/win32` provides backward compatibility.

Linking to System Libraries

MATLAB now links with the system libraries by default. You no longer need to specify them explicitly.

COM Automation Server Now Displays Figure

When using MATLAB as an Automation server, executing MATLAB commands that create figures now displays the figure window.

Previous releases of MATLAB created the figure in the background. To duplicate the old behavior, create a figure with its `Visible` property set to `off`, then set the property to `on` when you want the figure to be visible:

```
h = actxserver('matlab.application');
h.Execute('figure visible off');
h.Execute('plot(1:10)');
h.Execute('set(gcf,''visible'',''on'')');
```

R14SP2

Version: 7.0.4

New Features

Compatibility Considerations

Desktop Tools and Development Environment

Installation Folder with Spaces

In MATLAB 7.0.4 (R14SP2) software, the following two changes have been made to the MathWorks Installer on Microsoft Windows platforms:

- The Installer now allows a folder name with spaces in the installation path.
- The Installer honors the default installation folder for Windows software, which on most machines is `Program Files`.

These changes were made in response to many customer requests and the desire to conform to a widely established industry practice for the PC platform.

Note MathWorks products are used and integrated into many software environments. If you use MathWorks products in conjunction with other third party applications (compilers, other numerical analysis packages, etc.) you might want to continue installing into a folder that does not have spaces in the path until you have tested that those applications work with MathWorks products.

Startup and Shutdown

Confirmation Dialog Box for Quitting Added

When quitting MATLAB, a confirmation dialog box appears if you set a new preference for that purpose. By default, the confirm quitting preference is not set, so the dialog box will not appear. To change the preference, see the instructions for Confirmation Dialogs Preferences in the desktop documentation.

JVM Software Updated

MATLAB is now using version 1.5 of Sun Microsystems Java JVM software on Windows, Linus Torvalds Linux (32-bit), and Sun Microsystems Solaris platforms. Java software is supplied with MATLAB, so this change requires no action on your part.

Compatibility Considerations

If you use a specific version of Java with MATLAB on Windows, Linux 32-bit, or Solaris platforms, this change might affect you.

Desktop

Confirmation Dialog Boxes Preference Introduced

There are new preferences for displaying or not displaying confirmation dialog boxes for desktop tools. In previous versions, some of these preferences existed but were located with other preferences for the associated desktop tool. They are now organized in one preference panel for all desktop tools. Access them by selecting **File > Preferences > General > Confirmation Dialogs**. These preferences work in conjunction with the **Do not show this prompt again** check boxes that appears on various desktop confirmation dialog boxes. For more information, see Confirmation Dialogs Preferences in the desktop documentation.

Running Functions — Command Window and History

Overwrite Mode Now Supported

The Command Window now supports overwrite mode. Press the **Insert** key to enter text in overwrite mode. Press the **Insert** key again to return to entering text in insert mode. View the current state at the far right end of the status bar of the Command Window when it is undocked, or in the desktop when the Command Window is docked and has focus. In insert mode, **OVR** in the status bar is gray and the cursor has a wide block shape.

Hyperlink Color Preference Added

Set the color of hyperlinks that display in the Command Window. Select **File** > **Preferences** > **Command Window**, and under **Display**, select **Hyperlink color**.

Help

Subfunction Help Syntax Changed

To get help for a subfunction, use

```
help functionname>subfunctionname
```

Compatibility Considerations

In previous versions, the syntax was `help functionname/subfunctionname`. This change was introduced in R14 (MATLAB 7.0) but was not documented.

Bug Fixes and Known Problems Now on Web; No Longer Found Via Help Search

The Release Notes sections “Major Bug Fixes” and “Known Software and Documentation Problems” no longer include the content in the installed help files. Instead, the sections provide links to these lists on the MathWorks Web site. The lists on the Web site can be updated after the release date to reflect the latest information.

Compatibility Considerations

As a result of this change, the Help browser **Search** feature will not find search terms that are in the content of those reports. Use the MathWorks Web site search features to look for search terms in those reports.

Workspace, Search Path, and File Operations

Formatting Decimal Separator when Copying From the Array Editor

You can now specify how you want decimal numbers to be formatted when you cut or copy cells from the Array Editor and paste them into text files or other applications. You can specify a separator for this purpose in the `Array Editor` panel of the **Preferences** dialog. The **Decimal separator to use when copying** edit field is by default `"."` (period). If you are working in or providing data to a locale that uses a different character to delimit decimals, type that character in this edit field and click **OK** or **Apply**.

Workspace Browser Preference Panel Removed

The Workspace browser preferences panel was removed. The entry on that panel was for confirming deletion of variables. That preference is now part of **General > Confirmation Dialogs** preferences.

Compatibility Considerations

Use **File > Preferences > General > Confirmation Dialogs** instead of **File > Preferences > Workspace Browser**.

Current Directory Browser Preferences Added

There are new Current Directory browser preferences you can access by selecting **File > Preferences > Current Directory Browser, Browser display options**:

- View the file size by selecting the **Show file sizes** check box. (This is selected by default.)
- For models created with Simulink software, view brief descriptions in the **Description** column when **Show M and MDL file descriptions** is selected.
- For models created with Simulink, view the complete descriptions in the lower pane when the preference for **Show M, MDL and MAT file contents** is selected. This allows you to view information about a model without running Simulink.

Editing and Debugging M-Files

Go To Subfunction or Nested Function

Go directly to a subfunction or nested function within an M-file using the enhanced **Go To** dialog box. Access the dialog box by selected **Edit > Go To**. Click the **Name** column header to arrange the list of functions alphabetically, or click the **Line** column header to arrange the list by the position of the functions in the file.

Help Browser Now Accessible from MATLAB Stand-Alone Editor

You can now access the MATLAB Help browser from the MATLAB stand-alone Editor. This provides you with documentation for MATLAB, including using Editor features and MATLAB functions.

Preference for Editor/Debugger Dialog Moved

The **Show dialog prompt** preference has been moved to **Preferences > General > Confirmation Dialogs**. For more information, see Confirmation Dialogs Preferences in the desktop documentation.

Compatibility Considerations

Use **File > Preferences > General > Confirmation Dialogs** instead of **File > Preferences > Editor/Debugger** to set this preference.

Dragging Text Maintains Font and Highlighting

Now, when you drag text from the Editor/Debugger to another application, it maintains the syntax highlighting and font characteristics.

Source Control Interface

Register Project Feature Added; Add to Source Control Behavior Changed

There is a new source control interface feature for Windows platforms, **Register Project with MATLAB**. Use this to associate all files in a directory with a source control project. You perform this for any file in a directory, which registers the directory and all files in that directory. You only perform this once in a directory, and must perform it before you perform any other source control actions for files in that directory.

Access the feature in the Current Directory browser by right-clicking a file and selecting **Source Control > Register Your Source Control System Project with MATLAB** from the context menu. You can also access it from the Editor/Debugger **File** menu. To access the feature for files created with Simulink or Stateflow® software, use the Current Directory browser.

For a summary of the process, see the topic Source Control Interface on Microsoft Windows in the desktop documentation.

Compatibility Considerations

In previous releases, this feature was part of the **Add to Source Control** feature. You still need to add each file to source control, but you do this after first registering the directory that contains the file.

Project Name Exact Match No Longer Required

The name of the project in the source control system is no longer required to exactly match the name of the directory on disk containing the files.

Publishing Results

Cell Publishing: File Extension Changes

The files created when publishing using cells now have more natural extensions. JPEG files now have a `.jpg` instead of a `.jpeg` extension, and EPSC2 files now have an `.eps` instead of an `.epsc2` extension.

Compatibility Considerations

If you relied on the formerly used file extensions, you need to accommodate the changes.

Cell Publishing: LaTeX Image File Type Changes

Publishing to LaTeX now respects the image file type you specify in preferences rather than always using EPSC2 files.

Cell Publishing: Image Options More Restrictive

The **Publish image options** in Editor/Debugger preferences for **Publishing Images** have changed slightly. The changes prevent you from choosing invalid formats.

Notebook Support for Microsoft Word 97 Application to be Discontinued

Notebook will no longer support the Microsoft Word 97 application starting in the next release of MATLAB.

Compatibility Considerations

If you use Word 97 with Notebook, you will need to migrate to a more recent version.

Mathematics

New Vendor BLAS Used for Linear Algebra in MATLAB

MATLAB uses Basic Linear Algebra Subprograms (BLAS) for its vector inner product, matrix-vector product, matrix-matrix product, and triangular solvers in `\`. MATLAB also uses BLAS behind its core numerical linear algebra routines from Linear Algebra Package (LAPACK), which are used in functions like `chol`, `lu`, `qr`, and within the linear system solver `\`.

Starting in this release

- On Macintosh, MATLAB now uses the Accelerate framework.
- On 64-bit Linux, MATLAB uses Intel Math Kernel Library (MKL) 7.0.1 on Intel chips, and AMD Core Math Library (ACML) 2.0 on AMD chips.

max and min on Complex Integers Not Supported

Using the `max` and `min` functions on complex integer inputs (as shown in the example below) is no longer supported. This operation had been supported from release R11 through R14SP1, but now returns an error.

```
max(int8([3-4i 3+4i]))
```

Compatibility Considerations

Any code that calls `max` or `min` on complex integers should be removed from your program files.

Programming

Memory-Mapping

Memory-mapping is a mechanism that maps a portion of a file, or an entire file, on disk to a range of addresses within an application's address space. The application can then access files on disk in the same way it accesses dynamic memory. This makes file reads and writes faster in comparison with using functions such as `fread` and `fwrite`.

Another advantage of using memory-mapping in MATLAB is that it enables you to access file data using standard MATLAB indexing operations. Once you have mapped a file to memory, you can read the contents of that file using the same type of MATLAB statements used to read variables from the MATLAB workspace. The contents of the mapped file appear as if they were an array in the currently active workspace. You simply index into this array to read or write the desired data from the file.

Memory-mapped files also provide a mechanism for sharing data between applications. This is achieved by having each application map sections of the same file. This feature can be used to transfer large data sets between MATLAB and other applications.

textscan Enhancements

The `textscan` function originally read data only from files. As of this release, you can use `textscan` to read from strings as well.

xlsread Enhancements

In this release, you can write a function and pass a handle to this function to `xlsread`. When `xlsread` executes, it reads from the spreadsheet, executes your function on the data read from the spreadsheet, and returns the final results to you.

You can use either of the following syntaxes:

```
num = xlsread('filename', ..., functionhandle)
[num, txt, raw, X] = xlsread('filename', ..., functionhandle)
```

For an example, see the `xlsread` reference page.

xlsread Imported Date Format Changes

In MATLAB versions prior to R14, date values read into MATLAB from an Excel spreadsheet using `xlsread` were always imported as numeric date values. The R14 and later releases of MATLAB import dates in the format in which they were stored in the Excel file. Dates stored in string or date format are now imported as strings by `xlsread`. Dates stored in numeric format are imported as numeric date values.

Compatibility Considerations

Because of a difference in the way Excel and MATLAB compute numeric date values, any numeric dates imported from Excel into MATLAB must be converted to the MATLAB format before being used in the MATLAB application. For more information, see When to Convert Dates from Excel Files.

format Options Added

You can display MATLAB output using two new formats: `short eng` and `long eng`. See the `format` reference page for more information.

- `short eng` — Displays output in an engineering format that has at least 5 digits and a power that is a multiple of three.
- `long eng` — Displays output in an engineering format that has exactly 16 significant digits and a power that is a multiple of three.

```
format short eng
pi
ans =
     3.1416e+000

format long eng
pi
ans =
     3.14159265358979e+000
```

Nonscalar Arrays of Function Handles to Become Invalid

Creation of nonscalar arrays of function handles by `str2func` may be invalid or may return different results in future versions of MATLAB, but will continue to work in R14.

Compatibility Considerations

To avoid this warning and prepare for this change, convert the cell array of strings to a cell array of function handles.

For more information, type `help function_handle` and see the section entitled Note on Backward Compatibility.

Assigning Nonstructure Variables As Structures Displays Warning

Assigning to a nonstructure variable as if it were a structure is not recommended in MATLAB. For example, if variable `x` holds a double (as shown below), then attempting to add a field name to it, thus converting `x` to a structure, is not good programming practice and should generate an error.

```
x = 10;
x.name = magic(3);
```

Note that if `x` were empty (i.e., `x == []`), then assigning a field to it as if it were already a structure is acceptable.

Behavior Prior to Release R14

Because of a bug in releases of MATLAB prior to R14, you can assign a field to a nonempty, nonstructure variable in those releases without MATLAB generating a warning message or error. The result is that MATLAB quietly converts the variable to a structure:

```
x = 10;
class(x)
```

```
ans =
      double

x.name = magic(3);      % Invalid expression completes
                        %    without warning or error.
class(x)
ans =
      struct
```

Behavior In R14 and Later

In the MATLAB R14 and R14 service pack releases, you can still perform this type of operation, but MATLAB now displays a warning message:

```
x = 10;
x.name = magic(3);

Warning: Struct field assignment overwrites a value with class
"double".
```

In a future release of MATLAB, attempting this type of operation will throw an error instead of just displaying a warning message.

Compatibility Considerations

You are encouraged to modify any code that generates this warning. The section “Making a Valid Assignment” on page 35-10 gives instructions on how to do this.

Another Case — Extending the Depth of a Structure

The same rules apply when extending the depth of a structure by adding additional, lower-level fields. The first line of the example shown below creates a structure named `handle` and assigns to it a field of type double named `output`. The line after that treats this double as if it were a structure by attempting to assign a field named `time` to it. The second line is an invalid expression:

```
handle.output = 5;
handle.output.time = 13;
```

As in the case discussed earlier, this assignment does not generate a warning or error in MATLAB releases prior to R14. In the R14 and R14 service pack releases of MATLAB, you get the warning shown in the previous example. Beginning in a future release of MATLAB, this assignment will throw an error.

Making a Valid Assignment

To avoid this warning and future errors, first make x an empty structure or empty array as shown here. Once a variable is established as a structure or empty array, you can assign fields to it without getting an error:

```
x = struct;     or     x = [];
x.name = magic(3);
```

In the case of extending the depth of an existing structure, you can perform this type of assignment without generating a warning or error using the `struct` function as shown here:

```
handle.output = struct('time', 13);
```

Function Declaration Compatibility with Pre-R14 M-Files

As of Release 14, the function definition line in a function M-file no longer requires commas separating output variables. However, because this syntax is not compatible with earlier releases, you should always include the comma separators when writing an M-file function that you intend to run on releases both earlier and later than Release 14.

Compatibility Considerations

See Comma Separators Not Required in Function Declaration in the Release 14 release notes.

Graphics and 3-D Visualization

imwrite Now Supports GIF Export

The `imwrite` function now supports exporting image data in Graphics Interchange Format (GIF).

Cannot Dock Figures on Macintosh

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

Compatibility Considerations

You cannot dock figure in the Desktop on the Macintosh

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.

Compatibility Considerations

You cannot use plotting tools on the Macintosh.

Not All Macintosh System Fonts Are Available

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

Compatibility Considerations

Some Macintosh fonts are not available in MATLAB

XDisplay Property Setable on Motif-Based Systems

You can specify the value of the figure `XDisplay` property only on systems using Motif-Based figure windows.

Compatibility Considerations

The figure `XDisplay` property is supported only on systems using Motif.

Creating Graphical User Interfaces (GUIs)

New Callbacks Chapter

The Creating Graphical User Interfaces documentation offers a new chapter, in draft form, that attempts to bring information regarding callbacks into one place. It introduces the concepts and mechanisms with which you work, and explains some basic techniques for programming your GUI's behavior. This chapter is not yet complete, but you may find it useful, even in its current state, particularly if you are new to creating GUIs.

Temporarily, this new chapter appears as Appendix A, “Working with Callbacks (Draft).” It contains some new information, but also duplicates information that can be found in various places throughout the rest of the book. In cases where information has not yet been included in the new chapter, links take you to the main part of the book.

External Interfaces/API

New File Archiving Functions and Functionality

In addition to being able to `zip` and `unzip` compressed file archives, MATLAB software now supports the following archiving functions:

- `gzip`/`gunzip` — Compress/uncompress files in gzip format.

  `gunzip` reads archives from both file systems and URLs.
- `tar`/`untar` — Compress/extract files in a tar-file.

  `untar` reads archives from both file systems and URLs.
- The `unzip` function can now also open a zip archive from a URL.