MATLAB Release Notes

The "MATLAB 6.5.1 Release Notes" on page 1-1 describe the changes introduced in the latest version of MATLAB. The following topics are discussed in these Release Notes:

The MATLAB Release Notes also provide information about the earlier versions of the product, in case you are upgrading from a version that was released prior to Release 12.1. If you are upgrading from a release earlier than Release 12.1, you should also see these sections:

Printing the Release Notes

If you would like to print the Release Notes, you can link to a PDF version.

Contents

MATLAB 6.5.1 Release Notes

1

2

MATLAB 6.1 Release Notes

3

MATLAB 6.0 Release Notes

4

1

MATLAB 6.5.1 Release Notes

New Features

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

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

MATLAB Interface to Generic DLLs

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

Documentation

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

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

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

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

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

Restrictions for This Release

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

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

Function and Data Type Names in Generic DLL Interface

Minor changes have been made to the naming of some functions and data types in the Generic DLL interface. If you are upgrading from the post-release 13 download of MATLAB, see "Function and Data Type Names in Generic DLL Interface" on page 1-22 of these release notes.

Relational Operators Work with int64, uint64

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

Reading HDF5 Files

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

This section includes this information:

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

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

Overview of HDF5 File Structure

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

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

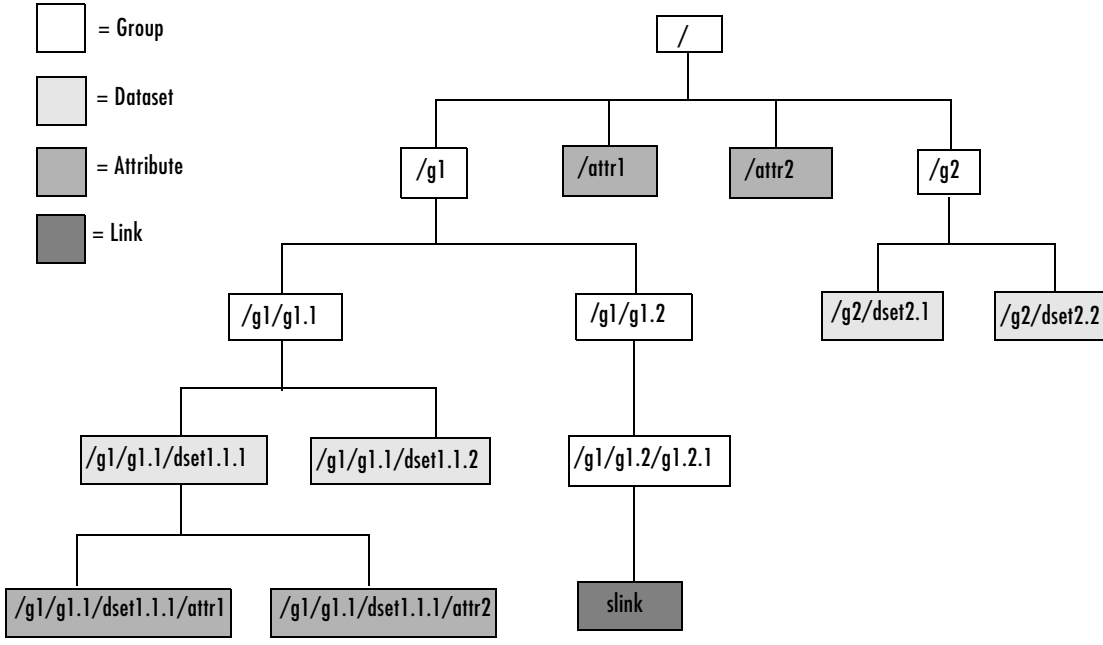


Figure 1-1: Hierarchical Structure of example.h5 HDF5 File

Determining the Contents of an HDF5 File

To extract an attribute or dataset from an HDF5 file, you must know the name of the attribute or dataset. You specify the name as an argument to the `hdf5read` function, described in "Reading Data from an HDF5 File" on page 1-6.

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

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

The `fileinfo` structure returned by `hdf5finfo` contains various information about the HDF5 file, including the name of the file and the version of the HDF5 library that MATLAB is using.

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

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

```
level1 = fileinfo.GroupHierarchy

level1 =

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

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

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

```
level2 = level1.Groups(2)

level2 =

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

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

```
dataset1 = level2.Datasets(1)

dataset1 =

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

Reading Data from an HDF5 File

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

name of a dataset, see “Determining the Contents of an HDF5 File” on page 1-5.

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

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

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

```
data =

  Columns 1 through 8

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000
1.6000    1.7000


  Columns 9 through 10

    1.8000    1.9000
```

Mapping HDF5 Data Types to MATLAB Data Types

The hdf5read function maps HDF5 data types to MATLAB data types, depending on whether the data in the dataset is in an *atomic* data type or a *non-atomic* data type.

HDF5 Atomic Data Types. If the data in the dataset is stored in one of the HDF5 atomic data types, `hdf5read` uses the equivalent MATLAB data type to represent the data. Each dataset contains a `Datatype` field that names the data type. For example, the dataset `/g2/dset2.2` in the sample HDF5 file includes this data type information.

```
dtype = dataset1.Datatype
dtype =

        Name: []
       Class: 'H5T_IEEE_F32BE'
    Elements: []
```

The `H5T_IEEE_F32BE` class name indicates the data is a four-byte, big-endian, IEEE floating point data type. (See the HDF5 specification for more information about atomic data types.)

HDF5 Non-Atomic Data Types. If the data in the dataset is stored in one of the HDF5 non-atomic data types, `hdf5read` represents the dataset in MATLAB as an object. To access the data in the dataset, you must access the `Data` field in the object.

To illustrate, this example uses `hdf5read` to read a dataset called `/dataset2` from the HDF5 file, `my_hdf5_file.h5`. The dataset contains four elements; each element is an HDF5 array.

```
data = hdf5read('my_hdf5_file.h5','/dataset2');
```

In MATLAB, the `hdf5read` function creates a a 1x4 array of `hdf5.h5array` objects to represent this data.

```
whos

Name    Size        Bytes     Class

data     1x4                hdf5.h5array

Grand total is 4 elements using 0 bytes
```

Index into the MATLAB array to view the first element in the dataset.

```
data(1)

hdf5.h5array:

Name: ''
Data: [4x5x3 int32]
```

To look at the raw data in the HDF5 array element, access the `Data` field in the object.

```
data(1).Data

ans(:,:,1) =
0 1 2 3 4
10 11 12 13 14
```

```
20 21 22 23 24
30 31 32 33 34

ans(:,:,2) =
100 101 102 103 104
110 111 112 113 114
120 121 122 123 124
130 131 132 133 134

ans(:,:,3) =
200 201 202 203 204
210 211 212 213 214
220 221 222 223 224 230 231 232 233 234
```

The `hdf5read` function uses any of the following objects to represent HDF5 non-atomic data types.

- `hdf5.h5array`
- `hdf5.h5enum`
- `hdf5.h5vlen`
- `hdf5.h5compound`
- `hdf5.h5string`

Reading and Writing Data with JPEG Lossless Compression

MATLAB now supports reading and writing data that has been compressed using JPEG lossless compression. With lossless compression, you can recover the original image from its compressed form. Lossless compression, however, achieves lower compression ratios than its counterpart, lossy compression.

Using the `imread` function, you can read data that has been compressed using JPEG lossless compression.

Using the `imwrite` function, you can write data to a JPEG file using lossless compression. For the `imwrite` function, you specify the `Mode` parameter with the `'lossless'` value.

Reading and Writing L*a*b* Color Data

The `imread` function can now read color data that uses the *L*a*b** color space from TIFF files. The TIFF files can contain *L*a*b** values that are in 8-bit or 16-bit CIELAB encodings or in 8-bit or 16-bit ICCLAB encodings.

If a file contains 8-bit or 16-bit CIELAB data, `imread` automatically converts the data into 8-bit or 16-bit ICCLAB encoding. The 8-bit or 16-bit CIELAB data cannot be represented as a MATLAB array because it contains a combination of signed and unsigned values.

The `imwrite` function can write *L*a*b** data to a file using either the 8-bit or 16-bit CIELAB encoding or the 8-bit or 16-bit ICCLAB encoding. You select the encoding by specifying the value of the `ColorSpace` parameter.

Major Bug Fixes

MATLAB 6.5.1 includes these major bug fixes:

Note In addition to the bug fixes described on this page, there are several bug fixes relating to MATLAB mathematics that are documented in a separate HTML bug-fix report.

Seeking Within a File

In Release 13, when you opened a file in write-only (`'wb'`) mode, you could not seed to a position in the file without first seeking to the beginning of the file. The `fseek` function has been fixed to allow seeking from any position of the file.

Reshaping to More Than Two Dimensions

In Release 13, under certain circumstances, reshaping an array to have more than two dimensions produced a two dimensional result. This has been corrected.

mkdir No Longer Fails On Windows NT

In Release 13, if on Windows NT you called the `dir`, `exist`, `isdir`, or `what` function on a nonexistent directory name on a network drive, it caused a windows handle to remain open to that directory name until you exit the MATLAB session. This condition caused any attempts to use `mkdir` on that directory to fail. This problem also affected the `mkdir` command when run from a DOS command prompt. This condition would persist until you exited MATLAB, thus freeing the handle.

This bug is fixed in this release.

Using sqrt with Complex Input

In Release 13, under certain circumstances, the `sqrt` function incorrectly produced a real result when called with a complex input. This bug has been corrected.

Multiplying Matrices with Non-Double Entries

In Release 13, MATLAB gave an incorrect answer or crashed for expressions of the following forms:

- `A' * B`
- `A * B'`
- `A' * B'`
- `A.' * B`
- `A * B.'`
- `A.' * B.'`
- `A' * B.'`
- `A.' * B'`

when either `A` or `B` was a numeric, non-double value (`single`, `int32`, etc.). This has been fixed for this release.

Sorting a Sparse Row Vector or Matrix

In Release 13, a segmentation violation occurred when you used the command `sort(S,2)` to sort a sparse row vector or to sort a sparse matrix along its rows. This bug is fixed in this release.

diff Produces Correct Results with Logical Inputs

In Release 13, the `diff` function could produce an incorrect result when you passed a logical array to it. This bug is fixed in this release.

Opening Modal Dialog with Third-Party GUI Open

In Release 13, MATLAB would occasionally hang if the user tried to open a modal dialog box when a third-party GUI was open. This no longer happens.

Serial Port Object with Latest Windows Service Pack

Under certain hardware configurations, or when using the latest Service Pack from Microsoft Windows, the serial port object in both MATLAB and the Instrument Control Toolbox could cause MATLAB to crash or hang. This problem is resolved in this release.

Several additional problems affecting the serial port have also been identified and fixed:

1. The serial port object now obeys all supported parity configurations.
2. The serial port object now obeys all supported flow control configurations.
3. On Windows, serial ports higher than `COM8` were not recognized by MATLAB. As of this release, MATLAB supports up to 256 ports.
4. The serial port object generates output empty events after running the serial port object continuously.

OpenGL Problem Using Notebook

This version of MATLAB uses an improved algorithm for selecting pixel formats when using the `UseGenericOpenGL` feature on Windows. This improvement fixes rendering problems seen with Notebook.

For information on graphics rendering, see Tech Note 1201.

Lcc C Compiler Fixed to Handle Large C Files

Lcc version 2.4.1 MathWorks patch 1.29 corrects a bug encountered when compiling very large C files. Although this bug has only been observed when

using large Stateflow models, we suggest that you upgrade to the new version to avoid potential problems when compiling MEX-files.

If you choose not to upgrade your version of Lcc, you can select a different C compiler using `mex -setup` from the MATLAB command line.

Bug Fixes in MATLAB Interface to COM

This release includes the following bug fixes in the COM interface:

Blank Spreadsheet Cells Returned as NaNs

When reading from a Microsoft Excel spreadsheet in a COM environment where MATLAB is the COM client and Excel the server, MATLAB now returns any empty cells in the spreadsheet as `NaN`s. In MATLAB 6.5 (Release 13), this same operation had returned a matrix of empty (`[]`) values.

For example, if the range `A1` to `D3` in a currently active workbook sheet contains no data, MATLAB 6.5.1 returns the following matrix of `NaN` values:

```
eActiveSheet = get(e, 'ActiveSheet');
eActiveSheetRange = get(eActiveSheet, 'Range', 'A1', 'D3');

eActiveSheetRange.Value
```

```
ans =
    [NaN]    [NaN]    [NaN]    [NaN]
    [NaN]    [NaN]    [NaN]    [NaN]
    [NaN]    [NaN]    [NaN]    [NaN]
```

Importing Excel Worksheets Containing Currency Format

In MATLAB 6.5, using a COM interface to Excel to import worksheet data containing currency format failed with either a field access error or segmentation violation. This bug is fixed in this release.

Getting the Forms Font Interface

In MATLAB 6.5, attempts to get the `Font` interface from a `forms.textbox.1` control, as done in the second line below, caused MATLAB to crash.

```
h=actxcontrol('forms.textbox.1')
font = h.Font
```

This bug is fixed in this release.

Programmatic Identifiers Containing Space Characters

Using the `actxcontrol` function with a `ProgID` argument containing one or more spaces failed in MATLAB 6.5. This bug is fixed in this release. For example, the following command now works:

```
h = actxcontrol('rmocx.RealPlayer G2 Control.1')
h =
   COM.rmocx.realplayer g2 control.1
```

Naming of Interfaces Returned by invoke or get

In MATLAB 6.5, interfaces returned by the `invoke` and `get` functions were given a name composed of the programmatic identifier (`ProgID`) for the component and the name of the method or property being invoked. In cases where a method or property implemented multiple interface types, this naming algorithm resulted in interface names that were not always unique.

For example, when invoking a method that returns an Excel and a Word interface, you could obtain any number of either type of interface (Excel or Word), but you could not obtain interfaces of both types. In such cases, you might be unable to access methods and properties of this interface.

In this release, interface names constructed by MATLAB are composed of the name of the type library and the class name, thus ending this potential naming conflict. If you invoke the method described in the last paragraph, MATLAB now returns the following for any Excel interfaces that you request:

```
Interface.Microsoft_Excel_9.0_Object_Library._Application
```

And MATLAB returns a different handle for Word interfaces:

```
Interface.Microsoft_Word_9.0_Object_Library._Application
```

Optional Input and Output Arguments Supported

MATLAB now supports optional input and output arguments passed in COM method calls. These arguments are declared as `[in, optional]` and `[out, optional]` respectively.

Memory Leak with MATLAB as COM Client

In Version 6.5, a memory leak developed under certain circumstances when MATLAB was configured as a COM client. This was caused by internal MATLAB code failing to release memory allocated by the method `StringFromCLSID`. This bug is fixed in this release.

Support for Multiple Type Libraries

MATLAB now supports multiple type libraries. If a COM object has many interfaces that are described in multiple type libraries, MATLAB can now retrieve the information correctly.

MATLAB Now Supports Skipping an Optional Argument

When calling ActiveX automation server methods, you can skip any optional arguments in the argument list by specifying that argument value as an empty matrix (`[]`). For example, the `Add` method shown below accepts as many as four optional arguments:

```
Add(Before, After, Count, Type)
```

To call this method, specifying values for `After` and `Count`, but no values for `Before` or `Type`, use this syntax.

```
addedsheet = invoke(Sheets, 'Add', [], Sheet1, 5);
```

Use [] for any arguments you skip, and that also precede the ones you do specify in the argument list. In this case, the Before argument is not specified but two subsequent arguments are.

Saving COM Objects Created with actxserver

Release 13 does not support saving COM objects that have been created with the actxserver function. You can use save only on control objects (created with actxcontrol). Attempting to use save on a COM server object causes MATLAB to hang temporarily, and eventually crash.

This bug has been fixed in this release so that if you now attempt to save a COM server object, MATLAB saves the object and any base properties of the object, but does not attempt to save any interfaces that might exist.

The same behavior applies to the pack function on COM objects.

This example creates a server running Microsoft Excel, adds a new property to the object, and saves it to the file excelserver.mat. It then reloads the server from the MAT-file.

```
e = actxserver ('Excel.Application');
addproperty(e, 'NewProperty');
set(e, 'NewProperty', 500);
get(e, 'NewProperty')
ans =
   500

save('excelserver.mat')
clear
get(e, 'NewProperty')
??? Undefined function or variable 'e'.

load('excelserver.mat')
get(e, 'NewProperty')
ans =
   500
```

Creating Certain Servers That Do Not Have Type Libraries

In the Release 12.1 and Release 13 releases, the actxserver function generated an error when attempting to create a COM object for certain servers. One error commonly returned by actxserver in these releases was

```
h = actxserver('msdev.application')
??? Error using ==> actxserver
Cannot find type library. COM object creation failed.
```

This has now been fixed in this release.

```
h = actxserver('msdev.application')
h =
   COM.msdev.application
```

Creating Microsoft Controls

Earlier versions of MATLAB would crash if you attempted to create certain Microsoft COM controls with the `actxcontrol` function. Examples of these controls, by programmatic identifier (`ProgID`), are shown below. MATLAB now successfully creates the controls.

```
mschart20lib.mschart     msdatalistlib.datacombo
msdatagridlib.datagrid   MSComCtl2.DTPicker.2
msdatalistlib.datalist   MSHierarchicalFlexGridLib.MSHFlexGrid.6
```

ActiveX Controls Created with Visual Basic 6.0

In Release 13, if you attach a callback routine to an event, and this event is eventually fired by a control created in Visual Basic 6.0, an error dialog box appears with the message “Run-Time error.”

This has been fixed in this release.

Type Mismatch Error Fixed

Some COM objects may define methods that pass scalar inputs by reference. This might appear in a type library signature as shown here for the `x` input:

```
functionname(double *x, [out] double *y)
```

Note that when input or output is not specifically stated, as is the case here for `x`, MATLAB defaults to input (`[in]`). So the line shown above is interpreted by MATLAB as

```
functionname([in] double *x, [out] double *y)
```

In MATLAB, the `[in]` and by-reference (`*`) specifications are considered incompatible for scalar arguments. In Release 13, MATLAB ignores the by-reference specifier for scalar inputs and passes such arguments by value

instead. Thus, any modified value for such an argument is not received by the calling function. You may also see a `type mismatch` error displayed, even when trying to access valid control methods.

MATLAB 6.5.1 fixes this bug by treating the `[in]` specifier for scalar references as if it were `[in,out]`.

In this example using MATLAB syntax, the `GetWinVersionX` function passes six `double` arguments by reference, yet none are returned in MATLAB 6.5:

```
GetWinVersionX = int32 GetWinVersionX(
   handle, double, double, double, double, double, double)
```

In MATLAB 6.5.1, all scalar reference arguments specified (or defaulting to) `[in]` are treated as `[in,out]`, and all references cause a value to be returned:

```
GetWinVersionX = [int32, double, double, double, double,
   double, double] GetWinVersionX(
   handle, double, double, double, double, double, double)
```

Note that this bug affects only scalar arguments. The `VT_DISPATCH` and `VT_VOID` types are not affected.

Bug Fixes in Creating GUIs

This release includes the following bug fixes related to creating, converting, and exporting GUIs:

Converting a MATLAB 5.3 GUI to MATLAB 6.5

Converting a MATLAB 5.3 (R11) GUI to MATLAB 6.5 sometimes resulted in the error:

```
Unhandled internal error in guidemfile. Reference to non-existent
field 'blocking'
```

This problem has been fixed.

Using GUIDE on Existing GUIs with Empty Tag Property

In MATLAB Version 6.5, editing a GUI that contained a uicontrol whose **Tag** property was set to `[]` (empty) sometimes generated the following error message:

```
Unhandled internal error in guidefunc.
Error using ==> set
Value must be a string
```

This problem has been fixed.

Exporting GUIs from GUIDE to a Single M-file

In MATLAB Version 6.5, some GUIs exported from GUIDE failed to open. In other cases, attempting to export a GUI resulted in one of the following errors:

```
??? Error using ==> guidefunc
Error using ==> ==
Matrix dimensions must agree.
```

```
??? Error using ==> guidefunc
Error using ==> ==
Function '==' is not defined for values of class 'struct'.
```

These problems have been fixed.

MATLAB Hangs when Using Property Inspector from GUIDE

Using the Property Inspector from GUIDE sometimes caused MATLAB Version 6.5 to hang. This problem has been fixed.

Recursion Limit Error when Running Existing GUIs from GUIDE

In MATLAB Version 6.5, running some existing GUIs from GUIDE generated the following error message:

```
??? Error using ==> guidefunc
Maximum recursion limit of 500 reached. Use
set(0,'RecursionLimit',N) to change the limit.  Be aware that
exceeding your available stack space can crash MATLAB and/or
your computer.
```

```
Could not create figure:
127
```

This problem has been fixed.

Upgrading from an Earlier Release

If you are upgrading from a release earlier than Release 13, see "Upgrading from an Earlier Release" on page 2-49 of MATLAB 6.5 Release Notes.

Rebuild Macintosh MEX-files

Macintosh MEX-files (named `.mex`) built with MATLAB 5.2 or older will not work with MATLAB 6.5 or later. You must recompile these files, creating a new file with the file extension `.mexmac`.

Function and Data Type Names in Generic DLL Interface

The following functions have been renamed since the initial download release of the Generic DLL Interface:

- The `libmethods` function is now called `libfunctions`.
- The `libmethodsview` function is now called `libfunctionsview`.

All data types ending in `Ref` are now suffixed with `Ptr`. For example, `doubleRef` is now called `doublePtr`, and `int16Ref` is now `int16Ptr`.

All data types ending in `RefPtr` are now suffixed with `PtrPtr`. For example, `doubleRefPtr` is now called `doublePtrPtr`, and `int16RefPtr` is now `int16PtrPtr`.

Known Software and Documentation Problems

For a list of bugs reported in the previous release that remain open, see "Known Software and Documentation Problems" on page 2-86 in the MATLAB 6.5 Release Notes.

Using xlsfinfo on Systems Without Excel

There is a bug in the `xlsfinfo` function that causes it to fail with the following error message when run on systems where Microsoft Excel is not installed.

```
Undefined function or variable 'xlsfinfo_old'.
```

We intend to fix this in the next release of MATLAB.

2

MATLAB 6.5 Release Notes

New Features

This section introduces the new features and enhancements added in MATLAB 6.5 since Version 6.1 (Release 12.1). This discussion of new MATLAB features is organized into the following categories:

If you are upgrading from a release earlier than Release 12.1, then you should also see "New Features" on page 3-2.

Development Environment Features

MATLAB 6.5 adds the following development environment features and enhancements.

JVM Version

On the Windows, Linux, Solaris, and Macintosh platforms, MATLAB uses Java Virtual Machine 1.3.1. Other platforms that support Java continue to use the JVM version they used for Release 12. To see the Java version that MATLAB uses, type

```
version -java
```

The HP-UX and IBM platforms do not support Java-based graphical user interfaces in MATLAB, and related products that rely on Java are not available on these platforms. See "Platform Limitations" on page 2-44 for details.

Startup

The toolbox path caching preference is on by default. This can result in significantly faster startup when MATLAB runs over a network or when you have many toolboxes. You will not see the improvement the first time you run MATLAB 6.5, but will after that. If you add or remove files and directories from `$matlabroot/toolbox` directories, you may need to update the cache. For details, see "Toolbox Path Caching" in the Development Environment documentation.

Desktop

Start Button. Click the **Start** button, 

, located in the lower left corner of the desktop, to readily access common MATLAB tools and features. It offers capabilities similar to those in the Launch Pad.

Status Bar. The status bar in the desktop now indicates the current state of MATLAB operations. For example, a `Busy` message appears while MATLAB is running an M-file.

New Profiler. Use the new Profiler graphical interface to assess your M-files so you can make changes to improve their performance. Select **View -> Profiler** from the desktop, or type `profile viewer`. The Profiler helps you take advantage of the new performance improvements that are part of the JIT Accelerator for MATLAB. For details, see "The Profiler" in the MATLAB Programming and Data Types documentation.

The new Profiler is based on the results returned by the `profile` function. You can still use the `profile` and `profreport` functions as you used them in Release 12.1.

Check for Updates. From the **Web** menu, select **Check for Updates**. A dialog box appears, listing the versions for all MathWorks products installed on your system. Click **Check for Updates** in the dialog box, which accesses the MathWorks Web site to determine if more recent versions are available.

Access MATLAB Central. From the **Web** menu, select **MATLAB Central** to access a page on the MathWorks Web site for exchanging M-files with other users and for accessing the `comp-sys.soft.matlab` Usenet newsgroup.

Change Current Directory. On UNIX platforms, you can now change the current directory field in the desktop toolbar using the **...** button to browse for the directory.

Apply Preferences. There is now an **Apply** button in the **Preferences** dialog box. When you click **Apply**, the preference change is made, but the dialog box remains open. This allows you to more easily experiment with changes to preferences.

Command Window

Find Feature. To find a term in the Command Window, select **Edit -> Find**. The **Find** dialog appears, in which you can enter a term and look for the previous or next occurrence.

Incremental Search. This is similar to the Emacs incremental search feature. In the Command Window, press **Ctrl+S** (or **Ctrl+Shift+S** for Windows key bindings) to display an incremental search field. Type a string in the field and the next occurrence of that string in the Command Window is highlighted. For more about this feature, see "Incremental Search" in the Development Environment documentation.

Hyperlinks to Run Functions. A new feature, `matlab:`, creates a hyperlink for specified text, which when clicked, runs the specified function. For example,

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

displays the link

Generate magic square

in the Command Window. When the user clicks the "`Generate magic square`" link, MATLAB runs `magic(4)`. Use this feature, for example, with the `disp` or `fprintf` functions.

Printing. You can now specify options for printing from the Command Window, such as including a header and printing line numbers. Select **File -> Page Setup** to set options. For more information, see "Page Setup Options for Printing" in the MATLAB documentation.

Preferences. There are new Command Window preferences for **Keyboard and Indenting**:

- **Command line key bindings**—Specify **Emacs (MATLAB standard)** or **Windows**. For example, with Emacs, **Ctrl+F** moves the cursor forward one character, whereas with Windows, **Ctrl+F** opens the **Find** dialog box.
- **Tab key**—These preferences previously existed on the general preferences tab for the Command Window.
- **Parentheses matching options**—MATLAB alerts you to matches and mismatches in pairs of delimiters, (that is, in parentheses ( ), brackets [ ], and braces { }), based upon MATLAB language syntax rules.

For details on the new preferences, see "Keyboard and Indenting Preferences" in the Development Environment documentation.

Open Selection. While in the Command Window, you can select text, right-click, and select **Open selection**. This runs the `open` function for the item you selected so that it opens in the appropriate tool. For example, you can open a variable in the Workspace browser, or open a file or function in the Editor. If no tool exists for the selected item, **Open selection** calls `edit`.

Command History

Printing. You can print the contents of the Command History and specify various printing options, such as including a header and printing line numbers. From the Command History window, select **File -> Page Setup** to set options. For more information, see "Page Setup Options for Printing" in the MATLAB documentation.

Find Feature. To find a term in the Command History, select **Edit -> Find**. The **Find** dialog appears, in which you can enter a term and look for the previous or next occurrence.

Autosave Command History. The Command History is automatically saved to a file on a regular basis. Specify options for what is saved and how often using Command History preferences. For details, see "Command History Preferences" in the Development Environment documentation.

Workspace Browser

You can rename a variable in the Workspace browser by right-clicking it and selecting **Rename** from the context menu. You can also select and copy variables in the Workspace browser, which puts their names (comma separated) onto the clipboard. You can then paste the names, for example, into the Command Window.

If you copy data from another application to the clipboard, use **Ctrl+V** in the Workspace browser to import the data to MATLAB using the Import Wizard.

Set Path

In the **Set Path** dialog box, you can now select multiple directories to remove from or to move within the path.

Current Directory Browser

Find M-Files Only. There are two new **Look in** options in the **Find** dialog box. Use them to limit the search in the current directory or in the entire MATLAB path to find only M-files.

Deleted Files to Recycle Bin. On Windows platforms, files you delete while using the Current Directory browser go to the Recycle Bin. You can by bypass the Recycle Bin by using **Shift+Delete**.

Change Current Directory. For UNIX platforms, you can now change the current directory field in the Current Directory browser toolbar using the **...** button to browse for the directory.

Changes Automatically Update Display. When you make changes to the current directory outside of MATLAB, the changes are automatically reflected in the Current Directory browser display. You do not have to select **Refresh** to show the changes.

File Operations

Following are functions that are new or have changed since the previous release. For more information, type `doc functionname`.

Function	New or Changed	Description
`copyfile`	Changed	The `writable` argument has been superseded by the `f` argument, although `writable` is still allowed for MATLAB 6.5. The function now also copies directories. It replaces the destination files or directories with the same name as the source files or directories without a warning—in previous versions, there was a warning in that event. If the destination files or directories are read-only and the `f` (or `writable`) argument is not used, `copyfile` will fail.
`fileattrib`	New	Set or get attributes of file or directory. The `fileattrib` function is like the DOS `attrib` command and the UNIX `chmod` command.
`mkdir`	Changed	Modified the message status—`mkdir` no longer returns `2` if the directory already exists, but instead displays a warning. It also has an enhanced return format.
`movefile`	New	Move file or directory. Can also be used to rename a file or directory.
`rmdir`	New	Remove directory, and optionally its contents as well.
`winopen`	New	For Windows users, allows you to open a file in the appropriate application, as if you double-clicked it in Windows Explorer.

Array Editor

Spreadsheet Behavior. You can now select and delete columns. You can cut, copy, paste, and delete cells. You can also exchange cells with Microsoft Excel via the operating system clipboard using these features.

You can set a preference to specify where the cursor moves to when you press **Enter**. For more about these enhancements, see the "Array Editor" in the Development Environment documentation.

Greater Number of Elements. The Array Editor can now show arrays with more than 10,000 elements. It does not support arrays with more than 65,536 (2^16) elements.

Editor/Debugger

Column Number, Line Number, and Current Function/Subfunction. In the Editor status bar, you can see the column number, line number, and function or subfunction for the current cursor location. When the Editor is docked in the desktop, the information appears in the desktop status bar.

Autosave Files. Files you change in the Editor are now automatically saved to a backup file on a regular basis. Use **File -> Preferences -> Editor/Debugger -> Autosave** to specify autosave options. For more information, see "Autosave" in the Editor/Debugger documentation.

Incremental Search. This is similar to the Emacs incremental search feature. Press **Ctrl+S** (or **Ctrl+Shift+S** for Windows key bindings) to display an incremental search field. Type a string in the field and the next occurrence of that string in the current file is highlighted. For more about this feature, see "Incremental Search" in the Editor/Debugger documentation.

Find Previous. You can find the previous occurrence of a string in a file by using **Edit -> Find Previous** after using any of the other **Edit -> Find** menu items.

Comment Formatting. You can specify a preference for the maximum width of comment lines and then apply that maximum to selected lines. You can also specify a preference to automatically wrap comment lines when they reach the maximum width. For more about this feature, see "M-file Comment Formatting" in the Editor/Debugger documentation.

Preferences for Parentheses Matching. There are new preferences for parentheses matching. The Editor/Debugger alerts you to matches and mismatches in pairs of delimiters, that is, in parentheses ( ), brackets [ ], and braces { }, based upon MATLAB language syntax rules. For more information, see "Parentheses Matching Preferences" in the Editor/Debugger documentation.

Printing Options. You can specify options for printing a file from the Editor, for example, including a header, by using **File -> Page Setup**. For more information, see "Page Setup Options for Printing" in the MATLAB documentation.

Invalid Breakpoints. When breakpoints are invalid, they appear gray instead of red. Breakpoints are invalid if there are unsaved changes or if there is a syntax error in the file. The breakpoints become valid when you save the file or when you fix the syntax error and save the file. For more information, see "Valid and Invalid Breakpoints" in the Editor/Debugger documentation.

Cannot Save in Debug Mode. You cannot save changes to an M-file while in debug mode. First quit debug mode and then save the file.

Integrated Text Editor Preference. If you install EmacsLink, a tool that allows you to debug M-files from the Emacs editor, you can specify an Editor/Debugger preference to use Emacs and EmacsLink for your default editing and debugging tools. For more information, see "Integrated Text Editor" in the Editor/Debugger documentation.

Wider and Resizable Line Number Column. You can view line numbers that contain up to nine digits. Drag the separator bar to the right of the line number column to make the column narrower or wider, allowing you to save space or see more digits.

Subfunctions Listed Alphabetically. When you click the function button  on the toolbar, the subfunctions are listed alphabetically. Previously they were listed in the order that they appeared in the M-file.

Open Selection. In an open M-file, the **Open Selection** feature has been enhanced. You can now jump from a subfunction call to the subfunction code within the current function M-file. To use this feature, select a subfunction call in an M-file, right-click, and select **Open Selection** from the context menu (or select **Open Selection** from the **File** menu). The Editor scrolls to that subfunction in the M-file. If that subfunction does not exist in the file, the open

function runs for the selected item, so that it opens in the appropriate tool. For example, you can open a variable in the Workspace browser, or open another file or function in the Editor. If no tool exists for the selected item, **Open selection** looks for a matching file in a private directory in the current directory.

Default Directory in Open Dialog Box and for New Files. The **Open** dialog box now opens to the MATLAB current directory. However, if you access the **Open** dialog box from the Editor, it opens to the directory for the current file in the Editor. When you create a new file, it is located in the MATLAB current directory.

Bookmark Support for All File Types. You can include bookmarks in any file type. Previously you could include bookmarks only in M-files.

New and Discontinued Features in edit Function. There is a new form of the `edit` function, `edit fun1 fun2 fun3 ...`, which opens all of the specified files in the default editor.

No longer supported are `edit fun1 in fun2` and `edit fun(a,b,c)`.

Save Not Available. Save is only available if a file has been changed. If there are no unsaved changes in a file, you can still use **Save As**, but you cannot use **Save**.

Help and Help Browser

Demos. To access demonstrations for all the MathWorks products you have installed, use the new **Demos** tab in the Help browser. For more information, see "Running Demonstrations" in the Development Environment documentation.

Boolean Operators in Search. In the Help browser **Search** pane, you can include the Boolean operators `AND`, `OR`, and `NOT` between terms you enter in the **Search** field. The operators must be in all capital letters and there must be a space before and after each operator.

For example, type `print OR printing AND figure NOT exporting` to find all pages that contain the words `print` and `figure`, or `printing` and `figure`, but only if the page does not contain the word `exporting`. At the top of the results list are any pages that contain all the `AND` and `OR` words in page titles.

Changes to Search Term Highlights. When you perform a search and select a resulting page to view, each word in the search term appears highlighted in a different color in the page. To clear the highlights, click the reload button  in the Help browser toolbar.

Setting the Documentation Location. You can now set the location where help files are stored (called the **Documentation location** in Help preferences) using the `docroot` function. You can include a `docroot` command in an M-file, such as a `startup.m` file.

Open Link in New Window. To open a linked page in a separate Help browser window, press **Alt** and right-click the link, or click the middle mouse button.

Visited Links. Visited links usually appear in a different color than unvisited links.

Print Range of Pages. When you print from the Help browser, the **Pages** field in the **Print** dialog box now shows the total number of printed pages required for the currently displayed page and lets you specify which of those pages to print.

Document Type Icons. Icons at the top two levels of the **Contents** pane indicate the type of document so you can quickly find a particular document type in the listing. For example, getting started documentation is represented by  (a green arrow), and function and blockset reference documentation are represented by  (orange pages).

Release Notes Location. Release notes for a product are now listed with that product in the **Contents** pane.

CD-ROMs for Documentation. For Windows platforms, there are now two CD-ROMS for documentation. To read PDF documentation from the CD-ROM, use the PDF Documentation CD.

Source Control

This release features expanded source control capabilities on PC platforms. You can interface to your source control system from by using MATLAB, Simulink, or Stateflow menus, or by using functions from the MATLAB Command Window.

The available source control system interface operations on PC platforms are

- Get latest version of file
- Add file
- Check out file
- Check in file
- Undo check out
- Remove file
- Show file version history
- Show file version differences
- Show file properties
- Start source control system

Notebook

Word Macros. Newer versions of Word have macro security features that might impact your use of Notebook. For details, see "Configuring Notebook" in the Development Environment documentation.

Word Versions Supported. Word for Office 2000 and 2002 (Office XP) are now supported. Word for Office 95 is no longer supported.

Using setup Option. The `setup` option is now easier to use. After running `notebook -setup`, you are prompted for your Windows version. The function performs all the remaining configuration with no additional input required from you. Only if the setup cannot find the files needed will you be prompted for additional information.

General

Demos. View and run demos via the new **Demos** pane in the Help browser. For platforms that do not support Java, run the `demo` command, which opens the Release 12.1 demo interface, and then follow instructions to access the demo files.

New perl Function. A new function, `perl`, calls the Perl script specified by the file `perlfile` using the appropriate Perl executable.

New Startup Option. There is a new startup option, `-logfile log`. It makes a copy of any output to the Command Window in the file `log`, including any crash reports.

Import/Export

New Functions for Exchanging Files with the Internet. MATLAB provides a set of functions for exchanging files with the Internet. These are URL, ZIP, and e-mail functions.

- Downloading URL content—From within MATLAB, you can read and save the content of a URL. The `urlread` function reads the content to a string variable in the MATLAB workspace. The `urlwrite` function saves the content to a file.
- ZIP functions—You can compress and uncompress files and directories from MATLAB using the `zip` and `unzip` functions.
- Sending e-mail—Use `sendmail` to send an electronic mail message, and optionally attachments, to a list of addresses.

File Format Support. The following table lists new import and export functions and highlights changes to existing functions.

Function	Purpose
`cdfepoch`	This new function converts a MATLAB date number or date string into the format supported by the Common Data Format (CDF).
`cdfwrite`	This new function supports writing data from MATLAB into Common Data Format (CDF) files.
`imfinfo`	The `imfinfo` function can now return information about Sun Raster image (RAS) files. In addition, when used with JPEG files, `imfinfo` now returns any comments that may be included in the graphics file. These comments are returned in the `comment` field as a cell array.

Function	Purpose (Continued)
`imformats`	This new function eases the task of adding read and write support for new file formats.
`imread`	The `imread` function can now read Sun Raster image (RAS) files and files over the internet.
`imwrite`	The `imwrite` function now supports two new formats: Sun Raster image (RAS) and PNM. PNM is not a file format but represents three other image formats, PBM, PGM, and PPM. When you specify PNM, `imwrite` chooses one of these three formats based on the contents of the data. In addition, `imwrite` supports the JPEG-specific parameter comment, in which you can specify any comment you want included in a JPEG file.
`multibandread`	This new function supports importing data from files that contain data divided into multiple bands, sometimes called raw files.
`multibandwrite`	This new function supports writing multidimensional arrays from MATLAB to a file in multiband format.

MATLAB HDF Import Tool. MATLAB 6.5 includes a new graphical user interface for importing data from an HDF or HDF-EOS files. This tool provides a graphical view of the data sets and metadata in an HDF file and lets you import selected data sets from the file by clicking a button.

Mathematics Features

MATLAB 6.5 adds the following mathematics features and enhancements:

These features are described below. At the end of this section are tables that summarize changes to the MATLAB math functions:

Delay Differential Equations

MATLAB now provides the capability to solve delay differential equations (DDEs) with constant delays. The DDE solver, `dde23`, provides an interface that is similar to the MATLAB ODE solver interface and is as easy to use. The supporting functions `ddeset`, `ddeget`, and `deval` enable you to set integration properties that affect problem solution and to evaluate the numerical solution obtained with `dde23`.

See "Initial Value Problems for DDEs" and the function descriptions in the MATLAB documentation for detailed information.

Singular Boundary Value ODE Problems

The function `bvp4c` can now solve a class of singular BVPs of the form

$$y' = \frac{1}{x}Sy + f(x, y)$$

$$0 = g(y(0), y(b))$$

posed on an interval $[0, b]$ with $b > 0$. The `bvpset` function provides a new `'SingularTerm'` integration property, which you can use to pass the constant matrix S to `bvp4c`.

See “Solving Singular BVPs” and the function descriptions in the MATLAB documentation for more information.

Integration Over a Volume

A new function, `triplequad`, evaluates a triple integral that you provide as a function, `fun(x,y,z)`, over a three dimensional rectangular region. As a default, `triplequad` uses the quadrature function `quad` to perform the integration. You can elect to use `quadl` instead or provide your own quadrature function.

Logarithmic Derivative of the Gamma Function

A new function `psi` evaluates the ψ function, also known as the digamma function, for each element of an array `X`. You can also use `psi` to evaluate the `k`th derivative of ψ, or a sequence of derivatives of different orders, at the elements of `X`.

Sparse, Square, Banded Matrix Left Division

Matrix left division (`\`) now uses banded solvers for `X = A\b` where `A` is sparse, square, and banded. Band density is defined as (# nonzeros in the band)/(# nonzeros in a full band). Band density = `1.0` if there are no zeros on any of the three diagonals.

If `A` is real and tridiagonal, i.e., band density = `1.0`, and `B` is real with only one column, `X` is computed quickly using Gaussian elimination without pivoting.

If the tridiagonal solver detects a need for pivoting, or if `A` or `B` is not real, or if `B` has more than one column, but `A` is banded with band density greater than the new `spparms` parameter `'bandden'` (default = `0.5`, in the interval `[0.0,1.0]`), then `X` is computed using LAPACK.

Sparse Matrix LU Factorization and Solve

LU factorization and solve for sparse matrices now uses UMFPACK. UMFPACK is a set of routines for solving unsymmetric sparse linear systems, $Ax = b$, using the Unsymmetric MultiFrontal method. It provides a considerable increase in computational speed for these matrices.

lu Function. The `lu` function provides two new syntaxes for sparse matrices. These new syntaxes use UMFPACK for factorization.

```
[L,U,P,Q] = lu(X)
[L,U,P,Q] = lu(X,thresh)
```

The syntaxes return a unit lower triangular matrix `L`, an upper triangular matrix `U`, and permutation matrices `P` and `Q`, so that `P*X*Q = L*U`. The `thresh` argument (default = `0.1`, in the interval `[0.0,1.0]`) controls pivoting.

\ (backslash). Matrix left division (`\`) uses UMFPACK for square sparse matrices that are not banded. You can control pivoting with the new `spparms` parameter `'piv_tol'` (default = `0.1`, in the interval `[0.0,1.0]`).

Information about UMFPACK is available online at `http://www.cise.ufl.edu/research/sparse/umfpack/`. The *UMFPACK Version 4.0 User Guide* is available at `http://www.cise.ufl.edu/research/sparse/umfpack/v4.0/UserGuide.pdf`. Type `help umfpack` at the command line for summary copyright and licensing information.

Matrix Math Performance Improvements for Triangular Matrices

The speed for solving linear systems `AX = B` where `A` is upper or lower triangular, and `B` is an `m`-by-`n` matrix, has been improved through the use of optimized Basic Linear Algebra Subroutines (BLAS). Optimized BLAS is provided by Automatically Tuned Linear Algebra Software (ATLAS).

BLAS has also been used to improve certain matrix multiplication operations, i.e., matrix*vector, vector*matrix, and rowvector*columnvector.

For the first time, ATLAS BLAS have been tuned to the Pentium 4 under both Windows and Linux operating systems, resulting in improved speed for core linear algebra functions.

By making better use of cache, the speed of matrix transposition has been increased for all matrices, but particularly for matrices whose size is a power of 2.

Summary of New Functions

Function	Purpose
`dde23`	Solve initial value problems for delay differential equations (DDEs) with constant delays
`ddeget`	Extract properties from the options structure created with `ddeset`
`ddeset`	Create/alter a DDE options structure that contains solver integration properties
`psi`	Psi (polygamma) function, i.e., the logarithmic derivative of the gamma function
`triplequad`	Numerically evaluate triple integral

Summary of Changed Functions

Function	Enhancement or Change
/ (slash) \ (backslash)	Now use banded solvers for sparse, square, banded matrices. See "Sparse, Square, Banded Matrix Left Division" on page 2-16 for more information. Now use UMFPACK for left and right division of square sparse matrices that are not banded. See "Sparse Matrix LU Factorization and Solve" on page 2-16 for more information.
/ (slash) \ (backslash)	The result of dividing a singular lower or upper triangular matrix by any other matrix, using either left (`\`) or right (`\`) division, may change. Previously, for singular square matrices `A` for which `rcond(A) = 0`, the result was always a matrix of `Infs`. This change is a result of the performance improvements described above. See "Mathematics Upgrade Issues" on page 2-51 for examples.

Function	Enhancement or Change (Continued)	
`bvp4c` `bvpset`	A new option `'SingularTerm'` enables you to specify a matrix as the singular term of singular BVPs. Set this option to the constant matrix S for equations of the form $$y' = S\frac{y}{x} + f(x, y, p)$$	
`corrcoef`	Provides three new syntaxes: `[R,P] = corrcoef(...)` returns `P`, a matrix of p-values for testing the hypothesis of no correlation. `[R,P,RLO,RUP] = corrcoef(...)` returns matrices `RLO` and `RUP` which contain lower and upper bounds for a 95% confidence interval for each coefficient. `[...]=corrcoef(...,'param1',val1,'param2',val2,...)` accepts parameter-value pairs that enable you to override the default confidence interval, and specify how to treat rows of `X` that contain `NaN`s.	
`deval`	Now also accepts output from `dde23`	
`gallery`	`house`	A new syntax `[v,beta,s] = gallery('house',x,k)` returns the norm of `x`. You can use the new argument `k` to control the sign of `s`.
	`leslie`	`gallery('leslie',a,b)` returns the `n`-by-`n` Leslie matrix with average birth numbers `a(1:n)` and survival rates `b(1:n-1)`.
	`orthog`	`gallery('orthog',n,k)` adds a new type, `k`, of matrix. k = 6 specifies a symmetric matrix arising as a discrete cosine transform such that `Q(i,j) = sqrt(2/n)*cos((i-1/2)*(j-1/2)*pi/n)`.

Function	Enhancement or Change (Continued)	
	`randsvd`	For large dimensions, a new argument, `method`, enables you to specify an alternative method that is much faster for large dimensions even though it uses more flops.
`legendre`	A new syntax `legendre(n,X,'norm')` computes the fully normalized associated Lengendre functions $N_n^m(x)$.	
`lsqr`	A new syntax `[x,flag,relres,iter,resvec,lsvec] = lsqr(...)` returns, in the vector `lsvec`, estimates of the scaled normal equations residual at each iteration.	

`lu`	Uses UMFPACK to improve speed for factorization of sparse matrices, and to add two new syntaxes for sparse matrices.
	`[L,U,P,Q] = lu(X)`
	`[L,U,P,Q] = lu(X,thresh)`
	The new output `Q` is the column permutation matrix that is used to reduce fill-in in the sparse case. `P` is the row permutation matrix that is used for numerical stability. The `thresh` argument controls pivoting. See "Sparse Matrix LU Factorization and Solve" on page 2-16 for information about UMFPACK.	

`qrdelete`
`qrinsert`
	Two new syntaxes for each function provide for the deletion and insertion of rows, as well as columns, in a QR factorization:
	`[Q1,R1] = qrdelete(Q,R,j,'col')`
	`[Q1,R1] = qrdelete(Q,R,j,'row')`
	`[Q1,R1] = qrinsert(Q,R,j,x,'col')`
	`[Q1,R1] = qrinsert(Q,R,j,x,'row')`
	The original syntaxes `qrdelete(Q,R,j)` and `qrinsert(Q,R,j,x)` default to `'col'`.	

Function	Enhancement or Change (Continued)	
`spparms`	Provides two new parameters for sparse matrix division.	
	`'piv_tol'`	Pivot tolerance used by the UMFPACK LU-based \ and /.
	`'bandden'`	Band density used by LAPACK-based \ and / for banded matrices.

Programming and Data Types Features

MATLAB 6.5 adds the following programming and data types features and enhancements:

JIT Accelerator with MATLAB

MATLAB 6.5 includes significant changes in the way MATLAB processes M-file functions and scripts. These changes affect the performance of MATLAB

and can give you a substantial performance increase over earlier MATLAB versions for many MATLAB applications.

Speeding up the execution of programs written in MATLAB is an ongoing MathWorks endeavor that will be delivered over a number of product releases. The documentation on "Performance Acceleration" explains how to best make use of the JIT Accelerator, how to use the MATLAB Profiler to optimize your performance, and includes several sample programs to illustrate how you can make your M-file programs run faster.

Regular Expression Support

MATLAB now supports searching and replacing characters using regular expressions. The following new functions support this capability. For more information, see "Regular Expressions" in the MATLAB documentation.

Function	Description
`regexp`	Match regular expression.
`regexpi`	Match regular expressions, ignoring case.
`regexprep`	Replace string using regular expression.

New Functions

MATLAB 6.5 added new functions for working with error generation, regular expressions, sorted sets, integer conversion, and for performing file operations.

Function	Description
`false`	False array
`fileattrib`	Set or get attributes of file or directory
`int64`	Convert to signed 64-bit integer
`isequalwithequalnans`	Determine if arrays are numerically equal, treating NaNs as equal
`issorted`	Determine if set elements are in sorted order

Function	Description (Continued)
`lasterror`	Return last error message and related information
`movefile`	Move file or directory
`orderfields`	Order fields of a structure array
`perl`	Call Perl script using appropriate operating system executable
`rethrow`	Reissue error
`rmdir`	Remove directory
`true`	True array
`uint64`	Convert to unsigned 64-bit integer
`winopen`	Open file in appropriate application (Windows only)
`xmlread`	Parse XML document and return Document Object Model node
`xmlwrite`	Serialize XML Document Object Model node
`xslt`	Transform XML document using XSLT engine

Cell to Matrix Conversion Functions

The following functions, previously belonging to the Neural Network Toolbox, are now core MATLAB functions.

Function	Description
`cell2mat`	Combine a cell array of matrices into one matrix
`mat2cell`	Break matrix up into a cell array of matrices

New Warning and Error Handling Features

These features include:

- "Formatted Error and Warning Strings"
- "Message Identifiers"
- "Warning Control Features"

Formatted Error and Warning Strings. Prior to MATLAB 6.5, the `error` and `warning` functions only accepted a simple string as an input argument, as shown here for `error`:

```
error('error_string')
```

In MATLAB 6.5, `error` and `warning` now accept a format string and one or more parameters, using a syntax similar to the MATLAB `sprintf` function. The syntax for `error` is shown here. Use the same syntax for `warning`:

```
error('format-string', arg1, arg2, ...)
```

Examples for using this syntax with `error` and `warning` are

```
error('File %s not found', filename);
warning('Ambiguous parameter name, "%s".', param)
```

Message Identifiers. A message identifier is a tag that you attach to an error or warning statement that makes that error or warning uniquely recognizable by MATLAB. You can use message identifiers with warnings to control any selected subset of the warnings in your programs, or with error reporting to better identify the source of an error.

Some examples of message identifiers are

```
MATLAB:divideByZero
Simulink:actionNotTaken
TechCorp:notFoundInPath
```

Message identifiers are used in warning control (see next section) and also to enable the `lasterr` and `lasterror` functions to better identify the source of an error.

See "Using Message Identifiers with lasterr" in the MATLAB documentation.

Warning Control Features. In this release, MATLAB gives you the ability to control what happens when a warning is encountered during M-file program execution. New options available in this release include

- Display selected warnings
- Ignore selected warnings

Depending on how you set up your warning controls, you can have these actions affect all warnings in your code, specific warnings that you select, or just the most recently invoked warning. See the section "Warning Control" in the MATLAB documentation for more information on this feature.

Also read the section, "Warning Control Upgrade Issues" on page 2-65 in these Release Notes to see how your existing programs might be affected by this change.

Dynamic Field Names for Structures

You can now reference structures using field names that are computed at run-time using the new *dynamic field names* feature in MATLAB. The dot-parentheses syntax shown below tells MATLAB to interpret `expression` as a dynamic field name:

```
structure_name.(expression)
```

This added capability now completes the following table by providing full dynamic access to all data types.

Data Type	Static (compile time)	Dynamic (run-time)
Matrix	`A(2,3)`	`A(m,n)`
Cell Array	`C{4}`	`C{k*2}`
Structure	`S.name`	`S.(field)`

See "Dynamic Field Names" in the MATLAB documentation for more information on this feature.

getfield and setfield Deprecated. Although the following two expressions are functionally equivalent, the first (using dynamic field names) offers improved execution speed and code readability. Compare the following for readability:

```
S(m,n).(fieldname)(k) = value

S = setfield(S,{m,n},fieldname,{k},value)
```

Since dynamic field names improve on the `getfield` and `setfield` functions, these two functions will eventually be removed from the MATLAB language. As of this release, `getfield` and `setfield` generate a warning message encouraging you to use dynamic field names instead.

New Logical AND and OR Operators for Short-Circuiting

Prior to this release, the MATLAB `&` (`AND`) and `|` (`OR`) operators served two purposes: that of a logical operator and also an array operator. These two roles at times conflicted, resulting in technically correct, yet possibly confusing evaluations.

MATLAB 6.5 introduces two additional `AND` and `OR` operators: `&&` and `||`. Use these new operators to evaluate a compound logical expression, especially when short-circuiting is required. Use the `&` and `|` operators for element-wise operations on arrays.

See "Short-circuit Operators" in the MATLAB documentation for a discussion on short-circuiting with `&&` and `||`.

New Output from ismember

The `ismember` function now returns an optional, second output indicating the indices at which members of a set are located. The syntax is

```
[tf, loc] = ismember(A,S,...)
```

When you use this syntax, `ismember` returns index vector `loc` containing the highest index in `S` for each element in `A` that is a member of `S`. For those elements of `A` that do not occur in `S`, `ismember` returns `0`.

For example,

```
a = reshape(1:5, [5 1])
set = [5 2 4 2 8 10 12 2 16 18 20 3];
[tf, index] = ismember(a, set);

index
index =
     0
     8
    12
     3
     1
```

New true and false Functions

MATLAB has two new functions for logical operations: `true` and `false`. `true` is shorthand for `logical(1)` and `false` for `logical(0)`. Both functions accept input arguments that enable you to build n-dimensional arrays of logical `1` or `0`.

This example builds a 3-by-5 array of type `logical`:

```
a = true(3,5)
a =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

`true(n)` is equivalent to `logical(ones(n))`, however `true(n)` is easier to use and will improve the readability of your program. Type `help true` or `help false` for more information.

Interrupting try-catch in a Loop

`try-catch` statements no longer catch **Ctl+C** interrupts. Prior to this release, pressing **Ctl+C** while a `try` block was executing would result in a jump to the corresponding `catch` block. In MATLAB 6.5, a **Ctl+C** aborts execution and returns control to the Command Window, regardless of what code is executing.

For example, in MATLAB 6.5, you can use **Ctl+C** to interrupt the loop shown here. You could not interrupt this loop in earlier MATLAB versions:

```
for k=1:100
   try
      pause(1);
   catch
   end
end
```

Changes to copyfile and mkdir

The `copyfile` and `mkdir` functions have changed since the previous release.

Changes to copyfile. The `writable` argument has been superseded by the `f` argument, although `writable` is still allowed for this release. The function now also copies directories. It replaces the destination files or directories of the same name as the source files or directories without a warning. (In previous versions, there was a warning in that event.) If the destination files or directories are read-only and the `f` (or `writable`) argument is not used, `copyfile` will fail.

Change to mkdir Return Status. `mkdir` no longer returns `2` if the directory already exists, but instead displays a warning. It also has an enhanced return format.

mfilename Returns Path and Class Information

You can now request more information from the `mfilename` function:

- `mfilename('fullpath')` — Returns the full path and name of the M-file in which the call occurs, not including the filename extension.
- `mfilename('class')` — In a method, returns the class of the method, not including the leading `@` sign. If called from a non-method, it yields the empty string.

Support for the 64-Bit Integers int64 and uint64

MATLAB now supports signed and unsigned 64-bit integers. Use the `int64` and `uint64` functions to convert a number to a signed or unsigned 64-bit integer.

64-Bit File Handling

MATLAB low-level file handling functions (`fopen`, `fseek`, `ftell`, etc.) now support 64-bit file offsets. This enables you to perform low-level I/O operations on files greater than 2 GB in size. (The limit in previous versions of MATLAB was `2^31-1` bytes, or 2 GB.)

64-bit support is available on the following platforms:

- Windows
- Solaris
- Alpha
- HPUX 11.0, 9000/785

64-bit support is *not* available on the following platforms, due to limitations imposed by their respective operating systems:

- SGI
- Linux
- HPUX 10.20, 9000/735
- HPUX 11.0, 9000/780

On the IBM-AIX platform, 64-bit file I/O is supported for reading only. You can write only up to 2 GB.

MATLAB Audio Enhancements

New audiodevinfo Function. On Windows 32-bit machines, `audiodevinfo` returns information about installed audio devices.

Enhancements to audiorecorder and audioplayer. `audioplayer` and `audiorecorder` can now take the audio device ID as input. You can obtain the device ID from `audiodevinfo`.

`audioplayer` can now take an `audiorecorder` as an input.

Support for 24-bit Recording and Playback. On 32-bit Windows machines with an installed 24-bit audio device, `audiorecorder` and `audioplayer` now support 24-bit recording and playback, respectively.

Improvement to wavread and wavwrite. `wavread` and `wavwrite` now support reading and writing 24- and 32-bit `.wav` files.

Workspace Browser Support. Right-clicking on an audio object in the Workspace Browser now displays a context menu with player/recorder controls.

New MATLAB Timer Object

MATLAB includes a timer object that you can use to schedule the execution of MATLAB commands. To use a timer, you must perform these steps:

1. Create a timer object by calling the `timer` function.
2. Specify which MATLAB commands you want executed and when you want them executed by setting timer object properties. (You can also set timer object properties when you create them, in Step 1.)
3. Start the timer by calling the `start` or `startat` functions.

Programming Tips Documentation

A number of questions come up repeatedly in our external customer newsgroup and our technical support Web site. Some of these questions arise when MATLAB users are unable to find the information they are looking for in the documentation. The "MATLAB Programming Tips" documentation is a new feature in MATLAB 6.5, designed to make it easier to find help on a wide range of topics. Many of the questions addressed by the tips documentation were taken from discussions in the MathWorks newsgroup and from the technical support site.

"MATLAB Programming Tips" is a chapter in the MATLAB "Programming and Data Types" documentation. It is a categorized compilation of tips, covering topics such as Debugging, Input/Output, Managing Memory, Optimizing for Speed, etc. Each item is relatively brief to help you to browse through them and find information that will be useful. Many of the tips include a link into the MATLAB documentation to give you more complete coverage of the topic.

Graphics Features

MATLAB 6.5 adds the following graphics features and enhancements.

New Text Properties – Control Text Background

Text objects have the following new properties.

Property	Purpose
`BackgroundColor`	Color of text extent rectangle
`EdgeColor`	Color of the rectangle edge
`LineStyle`	Style of the rectangle edge line
`LineWidth`	Width of the rectangle edge line
`Margin`	Increase the size of the rectangle by adding a margin to the text extent

Colormap Editor – Modify Colormaps Interactively

The colormap editor is a tool that enables you to modify the colormap of the current figure. See the `colormapeditor` function description for more information and an example.

Redesigned Property Editor

The Handle Graphics Property Editor and associated help have been redesigned.

Selecting a Printer From the MATLAB Command Line

In earlier versions of MATLAB, you could select a nondefault printer for graphics from the MATLAB command line on UNIX systems only. In MATLAB 6.5, you can do this on Windows systems as well. Specify the printer using the `-P` switch in the `print` command.

For example, to print Figure No. 3 to a printer called Calliope, type

```
print -f3 -PCalliope
```

If the printer name has spaces in it, put quotes around the `-P` option, as shown here.

```
print -f3 '-Pmy local printer'
```

Using a Network Print Server

On Windows NT, Windows 2000, and Windows XP systems, you can print to a network print server using the form shown here for a printer named trinity.

```
print -P\\PRINTERS\trinity
```

This form is not supported on Windows 98 or Windows ME. On these platforms, you can print to a network printer only if you install a network printer using the **Add Printer** dialogs. When installed in this manner, these network printers work without the use of the `\\server\printer` notation, as they look the same as local printers.

External Interfaces/API Features

MATLAB 6.5 adds the following external interface and API features and enhancements:

- "New MX and MEX Functions" on page 2-32
- "New COM Client Support Features" on page 2-34

New MX and MEX Functions

There are a number of new logical `mx` functions provided as part of changing `logical` from an attribute to a MATLAB class, several additional `mx` functions, and two new `mex` error handling functions.

New Logical Functions. This release introduces seven new C `mx` functions to use with `logicals`.

Function	Description
`mxCreateLogicalArray`	Create N-dimensional, `logical` `mxArray` initialized to `false`
`mxCreateLogicalMatrix`	Create two-dimensional, `logical` `mxArray` initialized to `false`

Function	Description (Continued)
`mxCreateLogicalScalar`	Create scalar, `logical` `mxArray` initialized to `false`
`mxCreateSparseLogicalMatrix`	Create unpopulated, two-dimensional, sparse, `logical` `mxArray`
`mxGetLogicals`	Get pointer to `logical` array data
`mxIsLogicalScalar`	`True` if scalar `mxArray` of class `mxLOGICAL`
`mxIsLogicalScalarTrue`	`True` if scalar `mxArray` of class `mxLOGICAL` is true

Obsolete Logical Functions. The following two functions are now obsolete. Support for these functions will be removed in a future release.

Function	Description
`mxClearLogical`	Convert `mxArray` to numeric type
`mxSetLogical`	Convert `mxArray` to `logical` type

New mx Functions in C API. MATLAB 6.5 also introduces these new C `mx` functions.

Function	Description
`mxGetChars`	Get pointer to character array data
`mxCreateDoubleScalar`	Create scalar, double-precision array initialized to the specified value

Note `mxCreateDoubleScalar` replaces `mxCreateScalarDouble`, although the latter function is still supported at this time.

New mex Functions for Error Handling. There are two new C and Fortran MEX functions that enable you to specify a message identifier and message string when reporting an error or warning. These functions also accept formatting conversion characters, such as those used with the MATLAB `sprintf` function, in the error or warning message string.

Function	Description
`mexErrMsgIdAndTxt`	Issue error message with identifier and return to MATLAB prompt
`mexWarnMsgIdAndTxt`	Issue warning message with identifier

New COM Client Support Features

In an effort to provide a consistent interface to object-oriented technologies, MATLAB 6.5 introduces several changes that affect the way you interact with Component Object Model (COM) controls and servers through MATLAB.

Key benefits of this change are

- Robust memory management — Objects and interfaces are destroyed automatically when the variable that represents the object or interface is either reassigned or goes out of scope.
- Flexibility in event handling — Register and unregister a control's events with callback or event handler routines at any time after the control has been created.
- Custom properties — You can attach your own properties to a control and store any kind of data in the control.
- A graphical user interface for viewing and modifying COM properties.
- Multiple arguments with the `set` function.
- More useful information on methods returned by `invoke`.
- More detail in error messages.

See "Client Support for COM" in the MATLAB documentation for more information on these features. You may also want to read through the "External Interfaces/API Upgrade Issues" on page 2-76 to find out how these changes may affect your existing programs.

COM Demo. MATLAB includes three demos showing how to use the COM client interface. To run any of the demos, click on the **Demos** tab in the MATLAB Help Browser. Then click to expand the folder called Automation Client Interface (COM).

New MATLAB Functions for COM. There are a number of new functions available for the COM interface.

Function	Description
`addproperty`	Add custom property to COM object
`deleteproperty`	Remove custom property from COM object
`eventlisteners`	Return a list of events attached to listeners
`events`	Return a list of events that the control can trigger
`isevent`	Determine if an item is an event of a COM control
`ismethod`	Determine if an item is a method of a COM object
`isprop`	Determine if an item is a property of a COM object
`registerevent`	Register an event handler with a control's event
`unregisterallevents`	Unregister all events for a control
`unregisterevent`	Unregister an event handler with a control's event

MATLAB Functions New to COM. You can now use these MATLAB functions in the COM environment.

Function	Description
`fieldnames`	Return property names of a COM object
`inspect`	Display graphical interface to list and modify property values

Function	Description (Continued)
`methods`	List all methods for the control or server
`methodsview`	Display graphical interface to list method information

Specifying Property Names. You may abbreviate the names of properties, as long as you include enough letters in the name to make it unambiguous. Also, property names are not case-sensitive. For example, to get the value of the `OrganizationName` property from a COM server running an Excel application, you can use

```
get(h, 'org')
ans =
   The MathWorks, Inc.
```

Get and Set on Multiple Objects. You can use the `get` and `set` functions on more than one object at a time by putting the object handles into a vector and then operating on the vector. See "Get and Set on Multiple Objects" in the MATLAB documentation.

Enumerated Values for COM Properties. When setting the value for a COM property in MATLAB, you can now use an enumerated string in place of a numeric value. An enumerated string, such as `xlUnicodeText`, is much easier to remember than its equivalent numeric value, and thus makes it unnecessary to spend time looking up valid settings for a property.

To list all possible enumerated values for a property of object h, use

```
set(handle, 'propertyname')
```

To set a property to the value represented by an enumerated string, use this syntax. The `enumstring` argument can be abbreviated, as long as you use enough letters to make it unambiguous:

```
set(handle, 'propertyname', 'enumstring')
```

To get the current enumerated value of a property, use

```
get(handle, 'propertyname')
```

See "Using Enumerated Values for Properties" in the MATLAB documentation for more information.

Custom Properties. You can attach your own properties to a control using the `addproperty` function. The syntax shown here creates a custom property for control, `h`:

```
addproperty(handle, 'propertyname')
```

This example creates the `mwsamp` control, adds a new property called `Position` to it, and assigns the value `[200 120]` to that property:

```
h = actxcontrol('mwsamp.mwsampctrl.2', [200 120 200 200]);
addproperty(h, 'Position');
set(h, 'Position', [200 120]);
```

To remove custom properties from a control, use `deleteproperty` with the following syntax:

```
deleteproperty(h, 'propertyname')
```

See "Custom Properties" in the MATLAB documentation for more information.

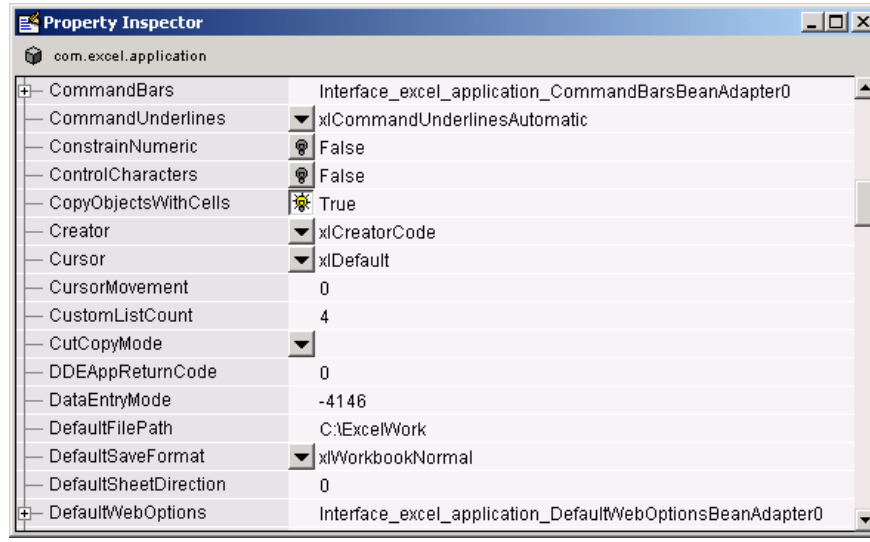
New Event Handling Functions. With earlier versions of MATLAB, you could register events for a control only at the time the control was created. In MATLAB 6.5, you can register and unregister events at any time using the `registerevent`, `unregisterevent`, and `unregisterallevents` functions.

You can also list all events that a control can respond to, or just those events that are currently registered, using the `events` and `eventlisteners` functions, respectively. The `events` function supersedes the COM `send` function.

See "COM Control Events" in the MATLAB documentation for more information on event handling.

GUI Interface to Get and Set Properties. Use the new `inspect` function to see a list of all properties belonging to a COM object or interface. Create an Excel server object and invoke `inspect` to bring up the Property Inspector window shown below:

```
h = actxserver('excel.application');
inspect(h)
```



To change the value of one of the properties, click on the property name at the left and then type in the new value in the field at the right.

See "Using the Property Inspector" in the MATLAB documentation.

set Accepts Multiple Arguments. You can now set more than one property value with one `set` command. The syntax is

```
set(h, property1, newvalue1, property2, newvalue2, ...);
```

Each `property` argument must be followed by a `newvalue` argument. The example shown here changes the `Label` and `Radius` for an `mwsamp` control:

```
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200]);
get(h)
ans =
     Label: 'Label'
    Radius: 20

set(h, 'label', 'Hello', 'radius', 35);
```

```
get(h)
ans =
     Label: 'Hello'
    Radius: 35
```

Returning More Than One Output Argument. A MATLAB client can now return more than one output argument from COM server applications. If you know that a server function supports multiple outputs, you can return any or all of those outputs to the MATLAB client.

With previous versions of MATLAB, you could only get back a single return value (shown here as `ret`) from a function call to the server, even for those functions that could return more than one output value:

```
ret = functionname(in1, in2, ...);
```

In MATLAB 6.5, you can specify additional output arguments (shown here as `out1 out2 ...`) in a function call, enabling the client access to all values returned by the function:

```
[ret out1 out2 ...] = functionname(in1, in2, ...);
```

If there are multiple output arguments, the return value is always the first argument on the left hand side (`lhs`).

MATLAB makes use of the pass by reference capabilities in COM to implement this feature. Note that pass by reference is a COM feature. It is not available in MATLAB at this time.

Argument Types Listed By Invoke. The `invoke` function now lists data types for input and output arguments. The function `m1` defined in an Interface Definition Language (IDL) file is shown here:

```
m1([in,out] BSTR* strInOut, [out] short* shortOut, [in] long
longIn, [out,retval] double* doubleRet);
```

MATLAB 6.1 `invoke` lists the function as

```
m1 = double m1(Variant(Pointer), Variant(Pointer), Int)
```

while MATLAB 6.5 `invoke` lists it as

```
m1 = [double, string, int16] m1(handle, string, int32)
```

More Detail in Error Messages. MATLAB now returns more detailed information when a COM error is generated. This includes the source and description of the error, along with the location of help resources provided to assist in resolving the error:

```
h = actxserver('excel.application');
Repeat(h)
??? Invoke Error, Dispatch Exception:
Source: Microsoft Excel
Description: Repeat method of Application class failed
Help File: D:\Applications\MSOffice\Office\1033\xlmain9.chm
Help Context ID: 0.
```

Creating Graphical User Interfaces (GUIDE) Features

MATLAB 6.5 adds the following features and enhancements to GUIDE:

- New structure for the generated M-file makes it easier to understand and program
- New GUIDE Quick Start dialog and GUI templates. When you first open GUIDE, the GUIDE Quick Start dialog box provides access to new GUI templates—simple examples of GUIs that you can modify for your own purposes.
- Tab Order Editor enables you to change the order in which GUI components are selected when a user clicks the **Tab** button.
- Changes to component tags automatically update callbacks and M-file code
- **Export** option in the **File** menu enables you to export a GUI to a single M-file that does not require a FIG-file.
- MATLAB Editor icon on the toolbar provides easier access to the editor.

Changes to the M-file Generated by GUIDE

The associated M-file generated by GUIDE has the following differences for Release 13:

- Most of the GUI initialization is performed in a separate function, which you do not need to edit.
- The GUI M-file contains an opening function, where you can add code to create data or perform other tasks before the GUI becomes visible to the user.

- The GUI M-file contains an output function for returning variables to the command line.
- A new calling syntax simplifies passing user defined arguments to the GUI M-file.
- You can pass figure property name/value pairs as arguments when creating the GUI.
- Generated callback function stubs no longer include a `varargin` argument. If you want to add more arguments to a callback subfunction, you must add the arguments to the function definition.

The following sections describe changes to the associated M-file in greater detail.

New Calling Syntax for GUI M-File. You can call the GUI M-file with the following syntax:

```
my_gui
my_gui('PropertyName', PropertyValue,...)
my_gui(UserArgs,...)
my_gui('PropertyName',PropertyValue,...,UserArgs,...)
```

- `my_gui` without arguments starts the GUI.
- Calling `my_gui('Property', Value,...)`, where `'Property'` is a valid figure property, creates a new `my_gui` using the given property value pairs.
- Calling `my_gui('My_function', hObject, eventdata, handles)` calls the subfunction `my_function` in the GUI M-file with the given input arguments.

Opening Function Code. The generated GUI M-file now includes a subfunction for any initialization code you want to execute. If you call the GUI with input arguments, they are passed to the opening function. The GUI M-file calls the opening function with the following arguments:

```
function myGUI_OpeningFcn(hObject, eventdata, handles, varargin)
```

- `hObject`—handle to figure
- `eventdata`—to be defined in a future version of MATLAB
- `handles`—structure with handles and user data (see `guidata`)
- `varargin`— command line arguments to my_gui (see `varargin`)

Output Function Code. The GUI M-file now includes a subfunction for passing output arguments to the command line. The GUI M-file calls the output function with the following arguments:

```
function varargout = myGUI_OutputFcn(hObject, eventdata, handles)
```

- `hObject`—handle to figure
- `eventdata`—to be defined in a future version of MATLAB
- `handles`—structure with handles and user data (see `guidata`)
- `varargin`—unrecognized property name/value pairs from the command line

GUIDE Quick Start Dialog and Templates

GUIDE now provides four templates that make it easier to construct GUIs. The templates are simple examples of GUIs that you can modify for your purposes. You can access the templates from the new GUIDE Quick Start dialog that appears when you open GUIDE, or when you select **New** from the file menu. It is often easier to build a GUI from an existing template rather than starting with a blank GUI.

openfig Accepts Property Name/Value Pair—Returns User Args

The `openfig` function enables you to specify figure property name/value pairs that are applied to the figure before it is displayed. See the `openfig` reference page for more information.

uiputfile and uigetfile Return Filter Index

The `uigetfile` and `uiputfile` functions can now optionally return an index value that enables you to determine which filter was selected by the user. See the `uigetfile` and `uiputfile` reference pages for more information.

uigetdir

The new `uigetdir` function displays a dialog box in which the user can select a directory, and returns the directory name as a string.

Major Bug Fixes

MATLAB 6.5 includes several bug fixes made since the last MATLAB release. This section describes the particularly important Version 6.5 bug fixes.

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

If you are upgrading from a release earlier than Release 12.1, then you should also see “Major Bug Fixes” on page 3-18.

Platform Limitations

The MATLAB functionality described in these Release Notes and in the MATLAB documentation applies to MATLAB 6.5, with the exception of the limitations listed below for the HP and IBM platform.

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

Another platform limitation involves the use BLAS on certain processors. See "You May Need to Overwrite the MATLAB Default Choice of BLAS" on page 2-47 for details.

Patch Required for HP-UX 11.0

To run MATLAB on HP-UX 11.0, you must install a patch available from Hewlett-Packard. To get the patch, go to `www.itrc.hp.com`, the IT Resource Center page. The patch is available to registered customers from the individual patches link. The patch name is below.

PHSS_21959 1.0 X/Motif 32 bit
Runtime 2000 Periodic Patch

Development Environment Limitations

The MATLAB 6.5 development environment features have the platform limitations described below. These include limitations that have existed since MATLAB 6.0.

The MATLAB desktop and most of the development environment tools are not available on the HP-UX and IBM platform. Following are the specific limitations for each tool and available alternatives.

Feature	Limitation and Alternatives
Desktop	Not supported. Instead, the MATLAB prompt appears in an X window. Use function alternatives for various tools.
Array Editor	Not supported. Instead, view and edit variables at the command line.
Command History	Not supported. To recall previous lines, use the up arrow key, or use the `diary` function or the `logfile` startup option.
Current Directory browser	Not supported. Use function alternatives documented for the Current Directory browser, including `cd`, `delete`, `ls`, and `mkdir`.
Demos	Demos for non-Java platforms run the way they did in Release 12.1. You do not access them from the Help browser, but rather by using the `demo` function.
Editor/ Debugger	Not supported. For editing M-files, use the `edit` function, and another text editor, such as Emacs—see the `edit` reference page to specify the other text editor. To debug M-files, use MATLAB debugging functions.
Help browser	Not supported. Help displays in your default browser. The Index and Search features are not available. You get a broken link message from your browser if you try to access documentation that you do not have installed.
HDF Import Tool	Not supported. Use function alternatives documented in Using the HDF Import Tool.
Import Wizard	Not supported. Use import function equivalents for the various features.

Feature	Limitation and Alternatives (Continued)
Launch Pad	Not supported. Access documentation and demos using functions, such as `help` and `demo`.
Preferences	Not supported. Set location of help files using `docopt`.
Profiler	The new desktop Profiler is not supported. The Version 6.1 `profile` and `profreport` functions are supported.
Set Path dialog box	Not supported. Use the `path`, `addpath`, and `rmpath` functions instead.
Source Control menu items	Not supported. Use the `checkin`, `checkout`, `cmopts`, and `undocheckout` functions on UNIX platforms or the `verctrl` function on PC platforms instead.
Workspace browser	Not supported. Use `who`, `whos`, `save`, `load`, and `clear` functions instead.

Mathematics Limitations

The MATLAB 6.5 mathematics features have the platform limitation described below.

Basic Fitting Interface

The Basic Fitting interface is not supported. Instead, use curve fitting functions such as `polyfit` and `spline`. See also “Data Analysis and Statistics” in the MATLAB documentation for more information.

Graphics Limitations

The MATLAB 6.5 graphics features have the platform limitations described below.

Feature	Limitation and Alternatives
Data Statistics	Not supported.
Printing	Uses the Release 11 **Page Setup**, **Print Setup**, and **Print** dialog boxes. For information about these interfaces, see "Printing MATLAB Graphics" in the online MATLAB documentation.
Property Editor	Not supported. Similar graphical user interfaces provide access to figure, line and text objects. Use the `set` and `get` functions to modify Handle Graphics object properties.

Creating Graphical User Interfaces (GUIDE) Limitations

The MATLAB 6.5 GUIDE-related features have the platform limitations described below.

GUIDE is not supported on the following platforms:

- IBM_RS
- HPUX
- HP700

You May Need to Overwrite the MATLAB Default Choice of BLAS

On the PC, under both Linux and Windows operating systems, MATLAB determines at startup time what processor your computer has, for example Genuine Intel Pentium II, Pentium III, or AMD Athlon. MATLAB then automatically selects the most appropriate BLAS for your processor. The same is true on the SUN, where MATLAB distinguishes between UltraSPARCs and non-Ultra machines.

However, on the remaining platforms you get the default BLAS, which is usually targeted for a reasonably modern or common processor:

- ALPHA 21264
- HP700 PA-RISC1.1
- HPUX PA-RISC2.0
- IBM_RS Power3
- SGI R12000

If you have reason to believe that your processor is closer to another of the flavors of BLAS distributed with MATLAB, for example 21164 on the ALPHA or PA-RISC2.0 on the HP700, you might want to override the default choice of BLAS. Look in your `<MATLAB>/bin/$ARCH` directory for libraries beginning with `atlas_` to see your options.

Overriding the Default

The way to override the default choice is to set the environment variable `BLAS_VERSION` before invoking MATLAB. For example (in `csh`):

```
setenv BLAS_VERSION  atlas_21164.so
setenv LAPACK_VERBOSITY 1
matlab
```

The environment variable `LAPACK_VERBOSITY` simply confirms that your choice of BLAS is being loaded once you start up MATLAB.

Restoring the Default

If you would like to return to using the default provided by MATLAB, you may use the command (in `csh`)

```
unsetenv BLAS_VERSION
```

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from MATLAB 6.1 to Version 6.5. This discussion of new MATLAB upgrade issues is organized into the following categories:

If you are upgrading from a release earlier than Release 12.1, then you should see "Upgrading from an Earlier Release" on page 3-23.

Development Environment Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of development environment features, are discussed below.

Toolbox Path Caching Now On By Default

Toolbox path caching is now on by default—see "Startup" on page 2-3.

Release 13 Prerelease users might not see the toolbox path caching option on by default. To turn it on, select **File -> Preferences -> General**, set the **Enable toolbox path cache** check box, and click **OK**. The next time you start MATLAB, it will create the cache file, and startups after that will be faster.

Changes to ver Function

The `ver` function header now displays more detailed operating system output and the version, if any, of the Java Virtual Machine MATLAB uses. The `hostid` is no longer in the `ver` header.

The `ver` header displays when you run `ver` with an argument, for example, `ver('simulink')`. The header is not displayed when `ver` returns the results to a structure, for example, `simver = ver('simulink')`.

The ver output no longer includes a date column. The output is now ordered with MATLAB first, Simulink second, if installed, and then all other installed products in alphabetical order.

Migration of Files Used by Desktop Tools

Most files associated with desktop tools are maintained when you upgrade from Release 12.1 to Release 13. Specifically, preferences, the Command History, Help favorites, and current directory entries in the desktop toolbar and Current Directory browser lists are maintained. However, there may be some invalid current directory and favorites entries if the locations of Release 13 files are different from the locations of Release 12 files.

pathdef.m. If you want Release 13 to use your existing `pathdef.m` file, save it to another location outside of `$matlabroot` before installing Release 13, and then after installing, copy it back.

Editor/Debugger

Cannot Save in Debug Mode. You cannot save changes to an M-file while in debug mode. First quit debug mode and then save the file.

Subfunctions Listed Alphabetically. When you click the function button  on the toolbar, the subfunctions are listed alphabetically. Previously they were listed in the order that they appeared in the M-file.

Use Delete Instead of Clear. The **Edit -> Clear** menu item was removed. Use **Edit -> Delete** instead.

Discontinued Form of edit. The `edit` function no longer supports the forms `edit fun1 in fun2` or `edit fun(a, b, c)`.

Running Playshow Demos from the Command Line

To run playshow demos from the command line, you now need to type `playshow` followed by the demo name. In previous releases, you only needed to type the playshow demo name to run it.

For example, if you type `quake`, the demo does not run. View the H1 line for `quake.m`, that is, the first comment line. It begins with two comment symbols (`%%`), indicating that `quake` is a playshow demo. Therefore, type `playshow quake` to run the demo.

Mathematics Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of mathematics features, are discussed below.

Singular Triangular Matrix Division

The result of dividing a singular lower or upper triangular matrix by any other matrix, using either left (`\`) or right (`/`) division may change. Previously, for singular square matrices `A` for which `rcond(A) = 0`, the result was always a matrix of `Infs`.

This change is a result of performance improvements described in "Mathematics Features" on page 2-15.

Example 1.

In MATLAB Version 6.5,

```
A = [1 2 3;0 4 5;0 0 0];
b = [1;2;3];
A\b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 0.000000e+000.

ans =
   NaN
  -Inf
   Inf
```

Previously, the result was

```
[Inf
 Inf
 Inf]
```

Example 2.

In MATLAB 6.5, a zero matrix is treated as a singular triangular matrix.

```
[0 0;0 0] \ [0 0]'
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 0.000000e+000.

ans =
   NaN
   NaN
```

Previously the result was

```
[Inf
 Inf]
```

Programming and Data Types Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of programming and data types features, are discussed below. MATLAB 6.5 introduces important changes to the inner structure of MATLAB that may affect existing programs. These changes are

Maximum Name Length Changed for Variables, Functions, Files

Prior to this release, the length of MATLAB identifiers (variable names, function and subfunction names, structure fieldnames, M-file names, MEX-file names, and MDL-file names) was restricted to 31 characters. Names using more than 31 characters were either truncated by MATLAB or caused a warning or error to be generated.

In MATLAB 6.5, any of these names can be up to 63 characters long.

A new function, `namelengthmax`, returns the maximum length for MATLAB identifiers.

```
namelengthmax
ans =
    63
```

If you have MATLAB programs that hard-code the maximum identifier length as 31, you should replace these hard-coded limits with a call to `namelengthmax`.

If you use identifiers that exceed 63 characters, MATLAB issues a warning and truncates any characters beyond the 63rd.

Characters Beyond 31 Are No Longer Ignored. In previous versions of MATLAB, if you had two or more long identifiers in which the first 31 characters were identical, MATLAB ignored any characters beyond the thirty first and thus recognized only one of the identifiers. For example, these two Stateflow filenames

```
stateflow_modelname_with_40_characters_1.mdl
stateflow_modelname_with_40_characters_2.mdl
```

both appeared to MATLAB 6.1 as shown below, and MATLAB recognized only one of the files:

```
stateflow_modelname_with_40_cha.mdl
```

In MATLAB 6.5, with the maximum MDL-file name length increased to 63, MATLAB recognizes both files. You should be aware of this change, as it could possibly lead to unexpected behavior.

Warning for Identifiers Longer Than 31. In MATLAB 6.5, if you use an identifier that exceeds the previous limit of 31 characters, MATLAB can optionally generate a warning of the form:

```
<identifier> exceeds MATLAB's previous maximum name length limit
of 31 characters.
```

This warning is disabled by default. You can enable it by typing

```
warning on MATLAB:usinglongnames
```

Note Unlike most MATLAB warnings, you cannot enable this warning with the commands, `warning on` or `warning on all`. You must use the command shown above.

Warning for Identifiers Longer Than namelengthmax. If you specify an identifier that exceeds the new character limit, MATLAB generates the following warning:

```
<identifier> exceeds MATLAB's maximum name length of
<namelengthmax> characters and has been truncated to
<truncated_identifier>
```

This warning is enabled by default. You can disable it by typing the following command. However, we strongly encourage you to leave it enabled:

```
warning off MATLAB:namelengthmaxexceeded
```

MATLAB Toolbox Functions Updated. MATLAB toolbox functions, such as `isvarname`, have been updated in MATLAB 6.5 to make use of the `namelengthmax` function, and thus return the correct values.

Effect On P-Code and MEX-files. You should recompile the following files:

- Any P-Code files that contain identifiers longer than 31 characters.
- Any MEX-files that use the constant, `mxMAXNAME`.

The logical Attribute Is Now a Class

In previous versions of MATLAB, `logical` was an attribute of any numeric data type. This is illustrated in the following example (executed in MATLAB 6.1), where `b = a > 10` produces a `double array` with a `logical` attribute:

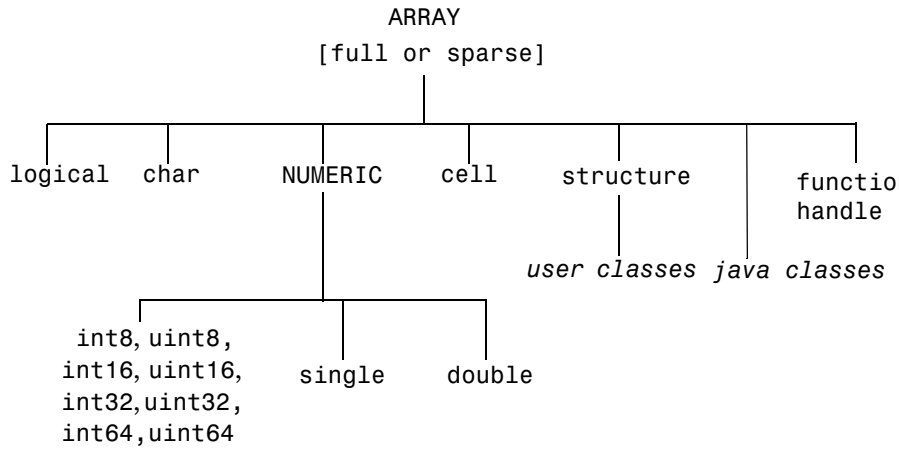
```
a = magic(4);
b = a > 10;

whos b
  Name      Size            Bytes  Class

  b         4x4               128  double array (logical)

Grand total is 16 elements using 128 bytes
```

In MATLAB 6.5, `logical` is a first class data type and MATLAB class. The class hierarchy diagram below shows `logical` to be a class, equivalent to other first class types like character and cell arrays.



The same example, executed in MATLAB 6.5, produces a result of class `logical array`:

```
a = magic(4);
b = a > 10;

whos b
  Name      Size                    Bytes  Class

  b         4x4                        16  logical array

Grand total is 16 elements using 16 bytes
```

Note Logical arrays in MATLAB 6.5 also require less storage space. In the example above, it took 128 bytes to store the array in MATLAB 6.1, and only 16 bytes in Version 6.5.

Effect on Related Functions. The table below compares the results obtained from a number of functions that operate on `logical` types. The variable a in the table is derived as follows:

```
a = (magic(4) > 10);
```

Command	MATLAB 6.1 Result	MATLAB 6.5 Result
`whos a`	`double array (logical)`	`logical array`
`class(a)`	`double`	`logical`
`islogical(a)`	`1`	`1`
`isnumeric(a)`	`1`	`0`
`isa(a,'double')`	`1`	`0`
`isa(a,'logical')`	`0`	`1`
`double(a)`	`double array (logical)`	`double array`

Valid logical Values. `logicals` can only have the values `0` and `1`. When you convert real finite values other than `0` or `1` to logical, MATLAB gives them a logical `1` value and issues a warning message:

```
x = logical(5)
Warning: Values other than 0 or 1 converted to logical 1
x =
     1
```

Behavior of islogical is Unchanged. Note in the table above that `islogical` continues to return `1` for a `logical` array, as it did in previous releases for arrays with a `logical` attribute.

Array Manipulation. All generic array manipulation functions (e.g., subscripting, reference, assignment, concatenation, `size`, `length`, `numel`, `ndims`, `permute`, `diag`, etc.) work as they do in MATLAB 6.0, subject to the behaviors described in this section.

Boolean Functions. All Boolean functions (e.g., `and`, `or`, `not`, `xor`, `any`, `all`) work as they do in MATLAB 6.0, subject to the behaviors described in this section.

MAT-Files. MAT-files created with earlier versions of MATLAB that contain `logical` arrays will load correctly in MATLAB 6.5. Values other than `0` or `1` will be converted to `1`'s. A `double` array with a `logical` attribute, when loaded in MATLAB 6.5, will be a `logical` array.

Mixed-Mode Arithmetic. Mixed-mode arithmetic (e.g., arithmetic involving a `logical` and a `double`) dispatches to the function registered for the non`logical` data type. The `logical` is converted to that type and the operation proceeds. This behavior is fully backward compatible with how MATLAB works in MATLAB 6.0.

Converting logical to double. You can use the `double` function to convert a `logical` array to a `double` array:

```
b = magic(4) > 10;
whos b
  Name      Size                    Bytes  Class

  b         4x4                        16  logical array

Grand total is 16 elements using 16 bytes

b = double(b);
whos b
  Name      Size                    Bytes  Class

  b         4x4                       128  double array

Grand total is 16 elements using 128 bytes
```

Indexed Assignment. As a rule, MATLAB data types are preserved on indexed assignment. This now holds true for `logical`, as it is now a MATLAB data type.

This example creates an empty array of type `logical`. The indexed assignment to `double` that follows, `(a(1) = 1)`, does not change the type to `double`. Its `logical` type is preserved:

```
a = logical([]);
whos a
  Name      Size                    Bytes  Class

  a         0x0                         0  logical array

Grand total is 0 elements using 0 bytes

a(1) = 1;
whos a
  Name      Size                    Bytes  Class

  a         1x1                         1  logical array

Grand total is 1 element using 1 bytes
```

Passing NaN or Complex to logical Functions. Attempting to pass `NaN` or complex values to an `if` or `while` statement, or to `and`, `or`, `not`, or `logical`, now consistently generates an error:

```
logical(NaN)
??? Error using ==> logical
NaN's cannot be converted to logicals.

not(2j)
??? Error using ==> not
Operands to NOT must not be complex.
```

Creating Logical Matrices with the sparse Function. Previously, when creating sparse logical matrices, the `sparse` function accumulated entries when it encountered repeated indices. For example,

```
A = sparse([1 1 1], 1, logical([1 0 1]))
A =
    (1,1) 2
```

In MATLAB 6.5, `sparse` now returns an error, because the only valid logical values are 0 and 1:

```
A = sparse([1 1 1], 1, logical([1 0 1]))
??? Error using ==> sparse
Repeated indices are not supported for sparse logical matrices.
```

Changes to Definition of "Truth"

As MATLAB has evolved, its definition of *truth* has become complicated and inconsistent. One goal of MATLAB 6.5 is to present a simple and self-consistent definition of truth that applies in all situations.

This change affects the following types of operations.

Comparing Empty with Empty or Scalar. MATLAB now returns an empty array (`[]`) when you compare two 0-by-0 empty arrays, or a 0-by-0 empty array with a scalar. This behavior also affects comparisons performed inside `if` and `while` statements.

Comparing two 0-by-0 empty arrays:

```
a = [];

b = (a == [])
b =
     []
```

Comparing a 0-by-0 empty array with a scalar:

```
b = (a == 5)
b =
     []
```

This behavior is now consistent with all other binary operators (e.g., >, <, ~=, +, -, .*, etc.).

Comparing Empty with Nonscalar. MATLAB now returns a `dimension mismatch` error when you compare a 0-by-0 empty array with a sized array:

```
a = [];

a == [1 2 3]
??? Error using ==> ==
Matrix dimensions must agree.
```

Using NaN with any or all. The `any` and `all` functions now ignore `NaN`. Thus `any` now returns `0` for a vector having `NaN`s as its only nonzero elements. The behavior of `all` is unaffected by this change, but it means that `any` and `all` now behave consistently with other reduction operators like `min` and `max`:

```
a = [0 0 NaN 0 NaN];
any(a)
ans =
     0
```

Interaction with Objects. In previous versions of MATLAB, passing a user-defined object as the argument to `if` or `while` caused the interpreter to call the object's `double` method (assuming it had one) in order to convert it to something whose *truth* could be determined. MATLAB 6.5 handles this situation by looking first for a `logical` method for the object and, upon failing to find one, calls its `double` method, if one exists.

If you have objects that will participate in truth evaluation, you should provide a `logical` method for those objects. The `logical` method must return `0` or `1` (`false` or `true`).

The sparse Class Is Now an Attribute

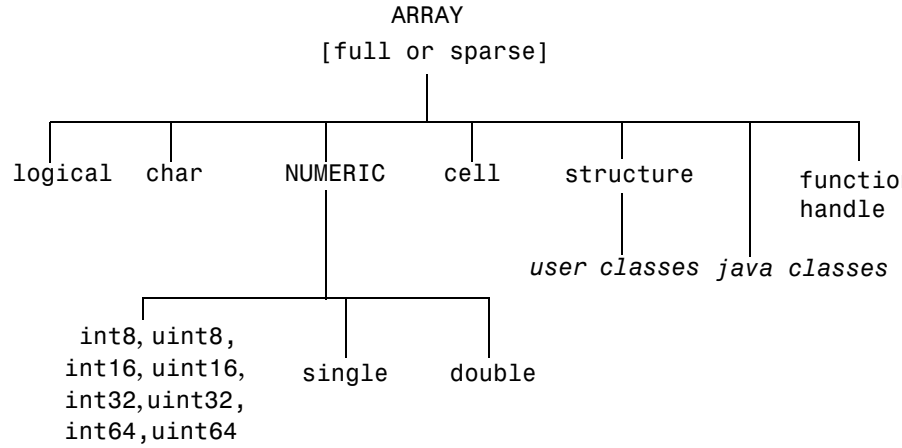
In previous versions of MATLAB, `sparse` was a first class data type and MATLAB class (subclass of `double`). This is illustrated in the following example (executed in MATLAB 6.1), where `sparse(eye(3))` produces a `sparse array`:

```
s = sparse(eye(3));
whos s
  Name      Size            Bytes  Class

  s         3x3                52  sparse array

Grand total is 3 elements using 52 bytes
```

In MATLAB 6.5, `sparse` becomes an attribute of a MATLAB class. The class hierarchy diagram below represents the MathWorks long-range plan for `sparse`, where `full` and `sparse` are attributes of all MATLAB classes.



Note In MATLAB 6.5, the `sparse` attribute is supported for the `double` and `logical` classes only.

The same example, executed in MATLAB 6.5, produces a `double array` with a `sparse` attribute:

```
s = sparse(eye(3));
whos s
  Name      Size                    Bytes  Class

  s         3x3                        52  double array (sparse)

Grand total is 3 elements using 52 bytes
```

Effect on Related Functions. The table below compares the results obtained from a number of functions that operate on sparse arrays. The variable a in the table is derived as follows:

```
a = sparse(eye(3));
```

Command	MATLAB 6.1 Result	MATLAB 6.5 Result
`whos a`	`sparse array`	`double array (sparse)`
`class(a)`	`sparse`	`double`
`issparse(a)`	`1`	`1`
`isa(a,'sparse')`	`1`	`0`
`isa(a,'double')`	`1`	`1`
`double(a)`	`double array`	`double array (sparse)`
`full(a)`	`double array`	`double array`
`x = logical(a)`	`sparse array (logical)`	`logical array (sparse)`
`class(x)`	`sparse`	`logical`
`full(x)`	`double array (logical)`	`logical array`

Note `issparse(a)` and `isa(a,'sparse')` give different results. The former indicates that `a` is a sparse matrix; the latter that `a` is not of the `sparse` class.

Behavior of issparse is Unchanged. Note in the table above that `issparse` continues to return `1` for arrays with a `sparse` attribute, as it did in previous releases for sparse arrays.

Arithmetic Operations. Arithmetic operations continue to work on sparse `doubles` as they do today.

Determining Storage Type. Make sure that your programs do not use `class` or `isa` to determine if a matrix uses sparse storage. Use `issparse` instead. It is both simpler and faster.

Methods in @sparse Directories. If you have written your own algorithms for dealing with sparse matrices and placed them in an `@sparse` directory, MATLAB will not access them because `sparse` is no longer a class.

MAT-Files. MAT-files created with earlier versions of MATLAB that contain sparse arrays will load correctly in MATLAB 6.5. A `sparse array`, when loaded in MATLAB 6.5, will be a `double` array with a `sparse` attribute, or a `logical` array with a `sparse` attribute if the array originally had the `logical` attribute. (See "The logical Attribute Is Now a Class" on page 2-55 for information on changes to `logical`).

Converting to Full. Use `full(x)` instead of `double(x)` to ensure that variable `x` is full. The `double` function no longer removes the sparseness of an array:

```
s = sparse(eye(3));
s = double(s);
whos s
  Name      Size                    Bytes  Class

  s         3x3                        52  double array (sparse)

Grand total is 3 elements using 52 bytes
```

Use `full` instead to make the array full:

```
s = full(s);
whos s
  Name      Size                    Bytes  Class

  s         3x3                        72  double array

Grand total is 9 elements using 72 bytes
```

Testing for full arrays. You should no longer use the following statement to test whether an array is full (nonsparse). Because the `sparse` class has been removed in MATLAB 6.5, this statement now returns `1` for both full and sparse arrays:

```
strcmp(class(s), 'double') == 1
```

In place of the above statement, use the following to test for a full array:

```
~issparse(s)
```

For example, create a sparse array, `s`, and test to see if it is a full array:

```
s = sparse(eye(3));

~issparse(s)
ans =
     0
```

Indexed Assignment. As a rule, MATLAB data attributes are not preserved on indexed assignment. This now holds true for `sparse`, as it is now an attribute.

This example creates a sparse `double` array. The indexed assignment to a full `double` that follows, (`s(:) = rand(3)`), removes the `sparse` attribute from the array:

```
s = sparse(eye(3));
whos s
  Name      Size                    Bytes  Class

  s         3x3                        52  double array (sparse)

Grand total is 3 elements using 52 bytes
```

```
s(:) = rand(3);
whos s
  Name      Size                    Bytes  Class

  a         3x3                        72  double array

Grand total is 9 elements using 72 bytes
```

Logical Indexing and find

The following two statements are intended to be equivalent in MATLAB. However, prior to this release, the statements were not equivalent in the case where `a` is non`double` and `b` contains only zeros:

```
a(find(b))
a(logical(b))
```

This has been fixed in this release. For the `a` and `b` shown here, the three equations that follow return the same result:

```
a = [int8(1) int8(2) int8(3)];
b = a > 5;

r1 = a(find(a > 5));
r2 = a(b);
r3 = a(a > 5);
```

As a result of this fix, you can now use either of the last two, simpler forms in place of the form that requires the use of `find`.

Warning Control Upgrade Issues

See the section on "Warning Control Features" on page 2-25 in these Release Notes for information on how you can control the way MATLAB handles the selected warnings in your programs.

Changes to Functionality. In some case, these changes to warning control will affect warning statements that currently exist in your code. The following two tables present how Versions 6.0 and 6.5 of MATLAB respond to MATLAB 6.0 warning syntax.

This table shows the MATLAB 6.0 behavior.

MATLAB 6.0 Syntax	MATLAB 6.0 Behavior
`warning backtrace`	`warning on all;` Enable backtraces.
`warning backtrace off`	`warning on all;` Disable backtraces.
`warning backtrace on`	`warning on all;` Enable backtraces.
`warning off backtrace`	`warning on all;` Disable backtraces.
`warning on backtrace`	`warning on all;` Enable backtraces.
`warning debug`	`warning on all; dbstop if warning`
`warning debug off`	`warning on all; dbclear if warning`
`warning debug on`	`warning on all; dbstop if warning`
`warning off debug`	`warning on all; dbclear if warning`
`warning on debug`	`warning on all; dbstop if warning`
`warning once`	`warning once ...` `<each-HG-back-compat-message-identifier>`
`warning always`	`warning always ...` `<each-HG-back-compat-message-identifier>`
`warning`	`ans` gets one of `on`, `off`, `debug`, or `backtrace`.
`s = warning(...)`	`s` gets one of `on`, `off`, `debug`, or `backtrace`. Process inputs (if any) as above.
`[s, f] = warning(...)`	`s` gets one of `on`, `off`, `debug`, or `backtrace`. `f` gets one of `once` or `always`. Process inputs (if any) as above.

For backward compatibility, MATLAB will continue to accept every usage of `warning` shown in the left column of the table above. However, some changes will be made to their actual behavior, as shown in the table below.

This table shows the MATLAB 6.5 behavior in response to MATLAB 6.0 `warning` command syntax.

MATLAB 6.0 Syntax	MATLAB 6.5 Behavior
`warning backtrace`	Enable backtraces.
`warning backtrace off`	Disable backtraces.
`warning backtrace on`	Enable backtraces.
`warning off backtrace`	Disable backtraces.
`warning on backtrace`	Enable backtraces.
`warning debug`	`dbstop if warning`
`warning debug off`	`dbclear if warning`
`warning debug on`	`dbstop if warning`
`warning off debug`	`dbclear if warning`
`warning on debug`	`dbstop if warning`
`warning once`	**`Warning`** - warning frequency is no longer supported.
`warning always`	**`Warning`** - warning frequency is no longer supported.
`warning`	`warning query all`; (doesn't assign to `ans`).
`s = warning(...)`	`s = warning('query', 'all');` then process inputs (if any) as above.
`[s, f] = warning(...)`	**`Warning`** - warning frequency is no longer supported.

Outputs Returned by Warning. Prior to MATLAB 6.5, `warning` returned up to two outputs: `state` and `frequency`. Neither of these is meaningful anymore, as there is no longer neither a single warning state nor a single warning frequency to return.

The `frequency` output is now disallowed. MATLAB generates a warning if you request this output.

The `warning` function now returns a structure instead of a string for the `state` output. Any existing code that uses this output should continue to function normally, but should be examined to make sure that the `state` value is properly interpreted in this new context.

Formatted Error and Warning Strings

For backward compatibility, if only one input is passed to `error` or `warning`, MATLAB treats it as a fixed string, not a format string. See "Formatted String Conversion" in Errors and Warnings in the MATLAB documentation.

getfield and setfield Superseded

Since dynamic field names improve on the `getfield` and `setfield`, these two functions will eventually be removed from the MATLAB language. In MATLAB 6.5, `getfield` and `setfield` will generate a warning message encouraging you to use dynamic field names instead. See the section, "Dynamic Field Names for Structures" on page 2-25 for more information on this.

New Behavior in break

The `break` function is intended to be used within a `for` or `while` loop. Use of `break` outside of a loop results in a warning being issued.

isequal and Structure Field Creation Order

When comparing structures with `isequal`, MATLAB no longer considers the order in which the fields of the structures were created in determining equality. See Example 2 on the `isequal` reference page.

Set Operations on Cell Arrays of Strings

The `intersect`, `setdiff`, and `setxor` functions have been modified to handle cell arrays of strings having one of more trailing spaces in a manner that is

consistent with its handling of other array types. These set functions no longer ignore trailing spaces when doing the comparison. See the examples below:

Prior to MATLAB 6.5	MATLAB 6.5

`intersect({'A'}, {'A  '})`
`ans =`
`  'A'`
	`intersect({'A'}, {'A  '})`
	`ans =`
	`  {}`

`setdiff({'A'}, {'A  '})`
`ans =`
`  {}`
	`setdiff({'A'}, {'A  '})`
	`ans =`
	`  'A'`

`setxor({'A'}, {'A  '})`
`ans =`
`  {}`
	`setxor({'A'}, {'A  '})`
	`ans =`
	`  'A'  'A  '`

Using mat2cell on an Empty Array

If you invoke `mat2cell` on an empty array, the function now returns an empty cell array rather than issuing an error. This requires that all zero dimensions of the empty input array have a corresponding `mat2cell` argument equal to `[]`.

In the following example, the third input argument to `mat2cell` specifies how MATLAB is to divide up the second dimension of the input array, `X`, in the resultant cell array. (See the `mat2cell` reference page for help on syntax.) Because the second dimension of `X` is of zero size, the only valid division specifier is `[]`.

```
X = rand(3, 0, 4);
C = mat2cell(X, [1 2], [], [2 1 1])
C =
   Empty cell array: 2-by-0-by-3
```

Concatenating with Empty Arrays

Empty arrays in concatenation operations can now affect the data type of the output. The only time you are likely to see this is when concatenating `doubles` and `logicals`. In previous versions of MATLAB, the example shown below returned a `double` array with a `logical` attribute. Now it returns a `double` because the empty `double` input is no longer ignored:

```
a = [ [] logical(0) ];
whos a
  Name      Size                    Bytes  Class

  a         1x1                         8  double array

Grand total is 1 element using 8 bytes
```

Concatenating Empty Cell Arrays. You cannot concatenate an empty cell array with numeric or character values:

```
a = ['string' {}]
??? Error using ==> horzcat
Conversion to cell from char is not possible.
```

Function Names Redefined As Variable Names

Under the conditions listed below, if you define a variable using a name that already belongs to a function, MATLAB issues the following warning message:

```
Variable <variable name> has been previously used as a function
name.
(Type "warning off MATLAB:mir_warning_variable_used_as_function"
to suppress this warning.)
```

This warning is issued only when all of the following conditions are true:

- The variable definition appears in an M-file function
- Within that M-file function, the name is used in a function call, and then later used as a variable name

For example, the first line of the function shown below uses the term `i` to call the MATLAB function that returns the complex constant. Some time later, in the `for` loop, the function code redefines `i` so that MATLAB now interprets it as a variable name, and assigns to the variable the values `1:10`.

```
function myfun
x = 5 + i;
```

```
for i = 1:10
   <do something>
end

y = 32 + i;
```

When MATLAB compiles this M-file function, it issues the warning message shown above. The reason for the warning is illustrated in the last line of the code shown in the example. The statement `32 + i` does not add the complex constant `i` to `32` as intended, but instead adds the value 10 to 32. This is because the opening `for` loop statement redefined the name `i` as a variable name and the last value assigned to that variable was `10`.

Note that this is a compile-time warning only. M-files run for the first time in a MATLAB session, or run after being cleared from memory (e.g., by `clear functions`) may issue this warning. M-files that are executed from cache do not.

Specifying More Outputs Than a Function Defines

A function call that requests more output values than are generated by the function being called now returns an error instead of a warning. Consider the function below that declares two outputs in the function definition line and assigns a value to one of them.

```
function [A, B] = mult_by_two(C)
A = 2 * C;
```

Calling this function with one output specified in the call completes successfully. Calling the function with two outputs specified returns an error. Even though two outputs are declared in the function definition line, only one output is generated in the function body.

```
[A, B] = mult_by_two(5)
??? One or more output arguments not assigned during call to
'mult_by_two'.
```

In previous versions of MATLAB, this type of call resulted in a warning. As a result, execution of the function continued and assignment to output variable `A` completed successfully.

In MATLAB 6.5, this type of call generates an error and aborts execution of the M-file. As a result, `A` remains undefined.

Consistent Handling of Subscripting Errors

The way in which MATLAB handles invalid subscripting is more consistent in MATLAB 6.5. MATLAB now responds to all of the situations listed below with this one error message:

```
??? Subscript indices must either be real positive integers or
    logicals.
```

The types of invalid subscripting that yield this error are shown in this table.

Type of Subscript	Example
Complex	`x(2i)`
Noninteger	`x(1.2)`
Negative	`x(-5)`
Zero	`x(0)`
`NaN`	`x(NaN)`
`Inf`	`x(Inf)`, `x(-Inf)`

The only functional change is that subscripting with noninteger values is now always treated as an error rather than a warning. In previous versions of MATLAB, this was an error only for sparse matrices.

Consistent Handling of Logical Errors

MATLAB now responds to the following types of invalid logical expressions as shown here:

- Bad arguments to `logical` (e.g., `logical(2)`):

  ```
  Warning: Logical was assigned values other than 0 or 1.
  ```

- Bad assignment to `logicals` (e.g., `x = logical([1 0 1]);  x(2) = 2`):

  ```
  Warning: Logical was assigned values other than 0 or 1.
  ```

- Complex assignment to `logical` (e.g., `logical(i)`):

  ```
  Complex argument is not allowed in LOGICAL.
  ```

- Using `NaN` in a logical expression (e.g., `logical(NaN)`):

```
NaN's cannot be assigned to logical arrays.
```

- `NaN` in an expression with `and`, `or`, `not` (e.g., `5 & NaN`):

```
NaN's cannot be converted to logicals.
```

Note Logical values are `1` (for `true`) and `0` (for `false`). Other nonzero values implicitly convert to `true`. Complex values and `NaN` cannot be converted implicitly. Use `~=0` to convert these to `logicals`. For example, `x = (NaN~=0).` to make `x` `logical`.

Empty Array in Comparisons. When you use an empty array in an equal or not equal comparison statement, MATLAB now returns an empty array as the result. In previous versions, MATLAB returned zero and displayed a warning.

For example, this statement now returns an empty array:

```
[] == 5
ans =
     []
```

The previous return value was a vestige of much older MATLAB behavior. The new return value is now consistent with all other binary operations involving empty arrays. For example, `[]` + 5 yields `[]`.

Operations That Are Now Considered Invalid

The following operations now return an error or warning.

Passing Complex to if or while. Passing a complex value to `if` or `while` now returns an error:

```
a = 5j;
if a
   disp 'true'
end
??? Complex values cannot be converted to logicals.
```

Previously, the imaginary part was ignored unless the argument was sparse.

Passing NaN to if or while. Passing `NaN` to `if` or `while` now returns an error:

```
a = NaN;
if a
   disp 'true'
end
??? NaN's cannot be converted to logicals.
```

Previously, this generated an error unless the `NaN` had the `logical` attribute.

Passing Complex to logical. Passing a complex argument to the `logical` function or assigning a complex value to a `logical` variable returns an error:

```
a = 5j;
x = logical(a);
??? Error using ==> logical
Complex values cannot be converted to logicals.
```

This has always been an error.

Passing NaN to logical. Passing a `NaN` argument to the `logical` function or assigning `NaN` to a `logical` variable now returns an error:

```
a = NaN;
x = logical(a)
??? Error using ==> logical
NaN's cannot be converted to logicals.
```

Previously, this assigned a the value `NaN` with a `logical` attribute.

Passing Complex to and, or, not. Passing a complex argument to the `and`, `or`, or `not` functions now returns an error:

```
a = 5j;
x = ~a;
??? Error using ==> ~
Operands to NOT must not be complex.
```

Previously, this assigned a the value zero.

Passing NaN to and, or, not. Passing `NaN` to the `and`, `or`, or `not` functions now returns an error:

```
a = NaN;
x = ~a
??? Error using ==> ~
NaN's cannot be converted to logicals.
```

Previously, this assigned a the value zero.

Assigning Nonlogical Values to logical. Passing incompatible arguments to the `logical` function or assigning them to a `logical` variable now generates a warning. Note in the example below that the value assigned to the `logical` array (`100`) is converted by MATLAB to `logical 1`.

```
a = (magic(4) > 10);
a(2,3) = 100
Warning: Values other than 0 or 1 converted to logical 1

a =
     1     0     0     1
     0     1     1     0
     0     0     0     1
     0     1     1     0
```

Previously, this resulted in no warning and assigned `100` to `a(2,3)`.

Graphics Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of graphics features, are discussed below.

Change to smooth3

Calculation of the `gaussian` filter option of the `smooth3` function has been corrected. This change may result in visual changes to graphs made with the smoothed data.

External Interfaces/API Upgrade Issues

The issues involved in upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of external interfaces and API features, are discussed below. These include

Changes to logical and sparse Data Types

In MATLAB 6.5, the `sparse` data type has been changed to be an attribute of its underlying data type. Also, the `logical` data attribute has been changed to be a first class data type. See "Programming and Data Types Upgrade Issues" on page 2-52 for more information on this change.

The following sections describe how this change may affect your C programs.

No Change to mxIsLogical. The `mxIsLogical` function is unchanged in MATLAB 6.5. It returns `true` for `logical` arrays, as it did for arrays with a `logical` attribute in previous releases.

Testing for Numeric. In previous releases, `mxIsNumeric` returned `true` for numeric arrays with the `logical` attribute. This function now returns `false` for `logical` arrays, since `logical` is a nonnumeric data type.

Using mxGetClassID on logicals. `mxGetClassID` returns a new `mxLOGICAL_CLASS` value for `logical` arrays.

Using mxGetClassID on Sparse Arrays. `mxGetClassID` no longer returns the enumerated value `mxSPARSE_CLASS`. Instead, it returns the enumerated value corresponding to the underlying data type. Use `mxIsSparse` to determine if an `mxArray` is sparse.

Testing for Sparse. In previous releases, you could use the following statement to determine if a matrix is sparse. This does not work in MATLAB 6.5.

```
mxGetClassID(x) == mxSPARSE_CLASS
```

You should use `mxIsSparse(x)` to determine if a matrix is sparse. The `mxIsSparse` function operates the same as in previous releases and also executes faster than the operation shown above.

Testing for Sparse with mxIsDouble. Because `sparse` has been changed from a MATLAB data type to a data attribute, `mxIsDouble(x)` no longer implies `~mxIsSparse(x)`, as it did in previous releases. Test the sparseness of an array using `mxIsSparse` instead.

No Change to mxIsSparse. The `mxIsSparse` function is unchanged in MATLAB 6.5. It returns `true` for arrays with a `sparse` attribute, as it did for `sparse` arrays in previous releases.

Obsolete logical Functions. The following two functions are now obsolete. Support for these functions will be removed in a future release.

Function	Description
`mxSetLogical`	Convert `mxArray` to `logical` type
`mxClearLogical`	Convert `mxArray` to numeric type

Functions Replaced in MATLAB 6.5

MATLAB handles `mxArrays` more efficiently in version 6.5 by not storing a variable name in the `mxArray`. When an `mxArray` name is required, these new C and Fortran functions enable you to pass it in the argument list.

The functions shown in the left column of the table replace those in the right column. The functions shown in the right column are now obsolete and may be unavailable in a future version of MATLAB.

New Function	Replaces
`mexGetVariable`	`mexGetArray`
`mexGetVariablePtr`	`mexGetArrayPtr`
`mexPutVariable`	`mexPutArray`
`engGetVariable`	`engGetArray`

New Function	Replaces
engPutVariable	engPutArray
matDeleteVariable	matDeleteArray
matGetVariable	matGetArray
matGetVariableInfo	matGetArrayHeader
matGetNextVariable	matGetNextArray
matGetNextVariableInfo	matGetNextArrayHeader
matPutVariable	matPutArray
matPutVariableAsGlobal	matPutArrayAsGlobal

For example, you should replace the second and third line shown here

```
parr = mxCreateDoubleMatrix(0, 0, 0);
mxSetName(parr, name);
retval = matPutArray(ph, parr);
```

with the second line shown below. The name of the `mxArray` is passed with `matPutVariable` rather than stored in the `mxArray` by `mxSetName`:

```
parr = mxCreateDoubleMatrix(0, 0, 0);
retval = matPutVariable(ph, name, parr);
```

mxCreateScalarDouble Replaced. New function `mxCreateDoubleScalar` replaces `mxCreateScalarDouble`. The latter function is still supported at this time, but support may be removed in a future release.

New Function	Replaces
mxCreateDoubleScalar	mxCreateScalarDouble

Compiling C++ Files

You no longer need to use the preconfigured options file, `cxxopts.sh`, to compile C++ MEX-files. MATLAB recognizes the following file extensions as C++ extensions, and automatically uses the C++ compiler.

```
.cxx
.cpp
.cc
```

The `cxxopts.sh` file is no longer available in MATLAB.

Also, on UNIX, you must now use the `-cxx` switch to the MEX script if you are linking C++ objects.

LCC Support for LAPACK

On Windows platforms, you can now compile and link C MEX-files that call LAPACK and BLAS functions using the MATLAB C compiler, `Lcc`. Use the following command to compile the file `myCmexFile.c` and link it with the LAPACK library file, `libmwlapack.lib`.

```
mex myCmexFile.c <matlab>/extern/lib/win32/lcc/libmwlapack.lib
```

The term `<matlab>` stands for the MATLAB root directory.

Client Support for COM

Client support for the MATLAB COM interface has changed significantly in MATLAB 6.5. There are many new features as well as important changes in previously supported features. This section describes how these changes may affect your existing programs.

See “New COM Client Support Features” on page 2-34 in these Release Notes and “MATLAB COM Client Support” in the MATLAB documentation for more information.

Creating an Object or Interface. When you create a COM control or server with `actxcontrol` or `actxserver`, MATLAB returns a COM object which now is displayed as `COM.<string>` rather than as `activex object`:

```
h = actxserver('Excel.Application')
h =
   COM.excel.application
```

This also applies to interfaces to a COM object. MATLAB represents the interface as `Interface.<string>` rather than as `activex object`:

```
w = get(h, 'Workbooks')
w =
   Interface.excel.application.Workbooks
```

New Error on Non-Existent ProgID. Both `actxcontrol` and `actxserver` return a different error message when an invalid `ProgID` is entered:

```
h = actxcontrol('xxxxx')
??? Error using ==> actxcontrol
Control creation failed. Invalid ProgID 'xxxxx'
```

Data Returned by get. Information returned by the `get` function now shows the type for each interface:

```
h = actxserver ('Excel.Application');
get(h)
ans =
      Application: [1x1 Interface.excel.application.Application]
           Parent: [1x1 Interface.excel.application.Parent]
          Windows: [1x1 Interface.excel.application.Windows]
        Workbooks: [1x1 Interface.excel.application.Workbooks]
              .
              .
```

Property names returned by `get` are no longer arranged alphabetically. They are displayed in the order that MATLAB gets them from the Type Library.

Set Invoked with No Arguments. When you invoke the `set` function without any arguments other than the object or interface handle, MATLAB no longer returns an error. Instead it returns a structure array, listing all properties for the object. The structure array also contains enumerated values for those properties that allow you to express values as enumerated strings.

Old error message:

```
set(h)
??? Index exceeds matrix dimensions.
```

Values returned in MATLAB 6.5:

```
set(h)
ans =
         Application: {}
             Creator: {'xlCreatorCode'}
              Parent: {}
              Cursor: {4x1 cell}
                .
                .
```

Change in Method Data Returned by invoke. The `invoke` function now returns more useful data on the methods of an object or interface. Note the differences shown in the example below:

```
h = actxserver('excel.application');

% Invoke, prior to MATLAB 6.5
invoke(h)
   DeleteCustomList = Void DeleteCustomList (Int)
   MailLogon = Void MailLogon (Variant[opt], Variant[opt],
      Variant[opt])
   NextLetter: 'Variant(Pointer) NextLetter ()'
         :
         :

% Invoke, in MATLAB 6.5
invoke(h)
   DeleteCustomList: 'void DeleteCustomList(handle, int32)'
   MailLogon: 'void MailLogon(handle, [Optional]Variant)'
   NextLetter: 'handle NextLetter(handle)'
         :
         :
```

Also note that the required `handle` argument is now explicitly shown.

Methods Function for COM. The `methods` function now returns the names for *all* methods of the specified class:

```
h = actxcontrol('MWSAMP.MWSampCtrl.1');
methods(h)

Methods for class COM.mwsamp.mwsampctrl.1:

AboutBox         GetR8Array         SetR8         move
Beep             GetR8Vector        SetR8Array    propedit
FireClickEvent   GetVariantArray    SetR8Vector   release
GetBSTR          GetVariantVector   addproperty   save
                                :
                                :
```

You can also use the `methodsview` function on COM objects now to get a graphical display of object properties.

Properties with Arguments. Any property that takes arguments is treated as a method in MATLAB 6.5. For example, the `Range` and `Item` properties in an Excel application server are now methods.

So, this statement, where `Item` is a *property* of `Sheets`

```
sheet2 = get(Sheets, 'Item', 2);
```

can now be replaced by the following, where `Item` is now a *method* of `Sheets`.

```
sheet2 = invoke(Sheets, 'Item', 2);
```

If you request a list of properties and methods for `Sheets` (using `get` and `invoke`, respectively), MATLAB now lists `Item` as a method.

For backward compatibility, functions in MATLAB 6.5 support properties that take arguments both as methods and as properties.

Arguments to Event Handlers. When a control triggers an event, MATLAB passes arguments from the control to any registered event handlers. MATLAB now passes two additional arguments:

- A string argument, holding the name of the event.
- A structure argument, holding the event name, control name, event identifier, event argument names, and event argument values.

See “Writing Event Handlers” in the MATLAB documentation for more information on changes affecting event handlers.

Specifying Events Using Identifiers. When registering events with their handler functions using either the `actxcontrol` or `registerevent` function, you can specify events either by event ID number or by event name.

Using event ID numbers:

```
h = actxcontrol('MWSAMP.MwsampCtrl.2', [0 0 200 200], f, ...
    {-600, 'myclick'; -601 'my2click'; -605 'mymoused'});
```

Using event names:

```
h = actxcontrol('MWSAMP.MwsampCtrl.2', [0 0 200 200], f, ...
    {'Click', 'myclick'; 'DblClick' 'my2click'; ...
     'MouseDown' 'mymoused'});
```

Use the new `events` function to display the names of all events recognized by the COM object in use. For example, to list all events for the `mwsamp` control, use

```
f = figure ('pos', [100 200 200 200]);
h = actxcontrol ('mwsamp.mwsampctrl.2', [0 0 200 200], f);

events(h)
   Click = void Click()
   DblClick = void DblClick()
   MouseDown = void MouseDown(int16 Button, int16 Shift,
      Variant x, Variant y)
```

Boolean Return Values. Invoking `get` on a property that returns a Boolean value now returns `1` to indicate `true`. Previously, it returned `-1` for `true`:

```
h = actxserver('Excel.Application');
set(h, 'DisplayStatusBar', 1);
get(h, 'DisplayStatusBar')
ans =
   1
```

Also, invoking a method that returns a Boolean value now returns `1` to indicate `true`. This also previously returned `-1` for `true`:

```
h = actxserver('Excel.Application');
invoke(h, 'Wait', 5)
ans =
   1
```

Argument Callouts in Error Messages. When a MATLAB client sends a command with an invalid argument to a COM server application, the server sends back an error message similar to that shown here, identifying the invalid argument. Be careful when interpreting the argument callout in this type of message.

```
PutFullMatrix(handle, 'a', 'base', 7, [5 8]);
??? Error: Type mismatch, argument 3.
```

In the `PutFullMatrix` command shown above, it is the fourth argument, `7`, that is invalid. (It is scalar and not the expected array data type.) However, the error message identifies the failing argument as `argument 3`.

This is because the COM server receives only the last four of the arguments shown in the MATLAB code. (The `handle` argument merely identifies the server. It does not get passed to the server). So the server sees `'a'` as the first argument, and the invalid argument, 7, as the third.

As another example, submitting the same command with the `invoke` function makes the invalid argument fifth in the MATLAB client code. Yet the server still identifies it as `argument 3` because neither of the first two arguments are seen by the server.

```
invoke(handle, 'PutFullMatrix', 'a', 'base', 7, [5 8]);
??? Error: Type mismatch, argument 3.
```

Releasing and Deleting Controls or Servers. This release addresses a potential memory leak in MATLAB. The leak was caused by the following:

- MATLAB did not completely clean up COM objects or interfaces without the explicit use of `release` or `delete`. For example, the following `clear` command did not release all memory used by the object. An explicit `release(h)` was required before the `clear`:

  ```
  h = actxcontrol ('MWSAMP.MwsampCtrl.1');
  clear all
  ```

- When a variable representing a COM object or interface was successfully assigned a new value, MATLAB did not release all of the memory originally allocated to the object or interface.
- When such a variable went out of scope, MATLAB did not release all of the memory originally allocated.

Explicit release or deletion of a COM object or interface is no longer necessary. MATLAB successfully clears the object or interface from memory when `clear` is invoked, or when the variable that represents the object or interface is either assigned a new value or goes out of scope.

Creating Graphical User Interfaces (GUIDE) Upgrade Issues

Upgrading from MATLAB 6.1 to MATLAB 6.5, in terms of GUIDE-related features, involves the following issue.

Using GUIDE Version 6.5 to Open GUIs Created in Versions 6.0 or 6.1

GUIDE generates a GUI's associated M-file with a new structure for Version 6.5. If you open a GUI that was created in Guide versions 6.0 or 6.1 using GUIDE version 6.5, the GUI will continue to function as it did previously. However, GUIDE does not update the existing code in the GUI's M-file to match the new style. If you add new components to the GUI in GUIDE version 6.5, GUIDE generates callbacks for the new components using the new M-code style, but leaves the original callbacks unchanged.

Known Software and Documentation Problems

This section includes a link to a description of known software and documentation problems in MATLAB 6.5.

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

For a list of bugs reported in the previous release that remain open, see "Known Software and Documentation Problems" on page 3-29.

3

MATLAB 6.1 Release Notes

New Features

This section introduces the new features and enhancements added in MATLAB 6.1 since MATLAB 6.0 (Release 12.0).

This section about new features is organized into the following subsections:

If you are upgrading from a release earlier than Release 12.1, then you should see "New Features" on page 4-2.

Development Environment Features

Command Window

MATLAB 6.1 includes two command window enhancements:

- You can set a preference for the command window to wrap lines. Input and output lines wrap to fit within the current width of the command window.
- If an error message appears when running an M-file, click on the underlined portion of the error message, or press **Ctrl+Enter**. The offending M-file opens in the Editor, scrolled to the line containing the error.

Help Browser

When you select documentation for the product filter, you can clear all currently selected products or select all products.

Editor/Debugger

The Editor/Debugger has the following enhancements:

- You can set bookmarks in M-files in the Editor/Debugger so that you can go directly to a particular line in the file. To set a bookmark, position the cursor at the line you want to bookmark, and then select **Set/Clear Bookmark** from the **Edit** menu.

  After setting bookmarks, you can go to the next or previous bookmark in a file. This allows you to go directly to a marked spot. Use the **Edit** menu items **Next Bookmark** and **Previous Bookmark** to navigate. Bookmarks are not saved when you close a file.
- You can include line numbers when printing files from the Editor/Debugger. To include line numbers, select **Preferences -> Editor/Debugger -> Printing**. Under **Print options**, check **Print line numbers**.
- You can use keyboard shortcuts to comment or uncomment a selection in the Editor/Debugger. The shortcuts are platform dependent and are listed with the menu items on the Editor/Debugger **Text** menu.
- In the **Find/Replace** dialog box, settings for **Match case**, **Whole word**, and **Wrap around** are remembered for the next MATLAB session.
- You can find the previous occurrence of a selection in the Editor/Debugger by pressing **Ctrl+Shift+F3**. You can also find the previous occurrence of a string you entered into the **Find & Replace** dialog box by pressing **Shift+F3**.
- When you move an arrow key over a token, for example, an opening parenthesis, (, the token and its match are briefly underlined. If there is no matching token, the token appears with a strike-through mark,  .
- When you run a file from the Editor/Debugger and the file is not in a directory on the search path or in the current directory, a dialog box appears presenting you with options that allow you to run the file. You can either change the current directory to the directory containing the file, or you can add to the search path the directory containing the file.

  If the file you want to run is already in a directory on the search path or in the current directory, the current directory remains as is and there are no actions you need to take.

- When you add a breakpoint to a file that is not in a directory on the search path or in the current directory, a dialog box appears presenting you with options that allow you to add the breakpoint. You can either change the current directory to the directory containing the file, or you can add to the search path the directory containing the file.

  If the file you want to run is already in a directory on the search path or in the current directory, the current directory remains as is and there are no actions you need to take.
- If you type `edit filename` and `filename` does not exist, a prompt appears asking if you want to create a new file. If you select **Yes**, a blank file titled `filename.m` is created in the Editor/Debugger. You can turn off this option in preferences for the Editor/Debugger.

Current Directory Browser

In the **Find/Replace** dialog box, settings for **Match case**, **Whole word**, and **Subdirectories** are remembered for the next MATLAB session.

Also, you can delete directories that are not empty. All contents of the directory will be deleted along with the directory.

Workspace Browser

You can select the column on which to sort in the Workspace browser, as well as reverse the sort order of any column. Click on a column heading to sort on that column. Click on the column heading again to reverse the sort order in that column. For example, to sort on **Size**, click the column heading once. To change from ascending to descending, click on the heading again.

Source Control

If you use Merant PVCS with MATLAB source control features, you no longer need to specify the project configuration file using `cmopts`. If you did specify it in previous releases, you do not have to remove it as MATLAB will ignore it.

General

The `computer` function now displays the endian byte ordering of the computer with the following form.

```
[str,maxsize,endian] = computer
```

Mathematics Features

Evaluation of Solutions to Differential Equation Problems

A new function, `deval`, enables you to evaluate the solution of a differential equation problem at a vector of points from the interval in which the problem was solved. `deval` uses, as input, the output structure `sol` of an initial value problem solver (`ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`) or the boundary value problem solver (`bvp4c`). A new ODE solver syntax returns the structure `sol`.

Additional Functions Use Qhull

These functions are now based on Qhull:

- `delaunay` — two-dimensional Delaunay triangulation
- `convhull` — two-dimensional convex hull

These functions call `delaunay` and therefore are now indirectly based on Qhull:

- `voronoi` — two-dimensional Voronoi diagrams
- `griddata` — data gridding and surface fitting

These functions are in addition to the Qhull-based functions introduced in MATLAB 6.0 (Release 12.0): `convhulln`, `delaunay3`, `delaunayn`, `griddata3`, `griddatan`, and `voronoin`.

Math Function Summary Tables

This section summarizes

- New math functions
- Functions with new or changed capabilities

Note See “Upgrading from an Earlier Release” on page 3-23 for information about obsolete functions.

New Math Functions

Function	Purpose
`deval`	Evaluate the solution of a differential equation problem using the output of `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`, or `bvp4c`.
`erfcinv`	Inverse complementary error function.
`tetramesh`	Tetrahedron mesh plot for use with `delaunayn`.
`triplot`	2-D triangular plot for use with `delaunay`.

Math Functions with New or Changed Capabilities

Function	Enhancement/Change
`bvpinit`	New syntax `solinit = bvpinit(sol,[anew bnew])` extrapolates a solution `sol` as an initial guess for solving a BVP on an extended interval. It can copy parameters from the previous iteration or let the user to provide new ones. For more information, see "Boundary Value Problems for ODEs" in the MATLAB documentation.
`bvpset`	New `Vectorized` option lets you pass to the solver `bvp4c` an array of column vectors. This allows `bvp4c` to reduce the number of function evaluations, and may significantly reduce solution time. For more information see "Boundary Value Problems for ODEs" in the MATLAB documentation.
`convhull`	New syntax `[K,a] = convhull(x,y)` returns the area `a` of the convex hull.
`convhulln`	New syntax `[K,v] = convhulln(X)` returns the volume `v` of the convex hull.

Math Functions with New or Changed Capabilities (Continued)

Function	Enhancement/Change
`numel`	New syntax `n = numel(A, varargin)` returns the number of subscripted elements, `n`, in `A(index1,index2,...,indexn)`, where `varargin` is a cell array whose elements are `index1`, `index2`, `...`, `indexn`.
`ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`	New syntax `sol = solver(odefun,[t0 tf],y0...)` returns a structure that you can use with the new function `deval` to evaluate the solution at any point on the interval `[t0,tf]`.
`polyeig`	New syntax `e = polyeig(A0,A1,..,Ap)` returns only the eigenvalues of the specified eigenvalue problem. Use `[X,e] = polyeig(A0,A1,...Ap)` if you also want the eigenvectors. This capability is available in MATLAB 6.0 (Release 12.0).
`ppval`	New syntax `ppval(xx,pp)` transposes the input arguments to enable you to use `ppval` with function functions.
`qz`	New syntax `[AA,BB,Q,Z,V,W] = qz(A,B)` returns `W`, the left generalized eigenvectors of `A` and `B`.
`reshape`	New syntax `reshape(A,...,[],...)` calculates the length of the dimension specified by the placeholder `[]`.
`svd`	Can now return only the first two outputs, `U` and `S`, where `S` is a diagonal matrix of the same dimension as the input argument `X`, and `U` is a unitary matrix.

Programming and Data Types Features

Partial Evaluation of Expressions

Within the context of an `if` or `while` expression, MATLAB does not necessarily evaluate all parts of a logical expression. In some cases, it is possible, and often advantageous, to determine whether an expression is true or false through only partial evaluation. This is sometimes referred to as *short-circuiting*.

For example, if `A` equals zero in statement 1 below, then the expression evaluates to `false`, regardless of the value of `B`. In this case, there is no need to evaluate `B` and MATLAB does not do so. In statement 2, if `A` is nonzero, then the expression is `true`, regardless of `B`. Again, MATLAB does not evaluate the latter part of the expression.

```
1)  if (A & B)        2)  if (A | B)
```

You can use this property to your advantage to cause MATLAB to evaluate a part of an expression only if a preceding part evaluates to the desired state.

Note Partial evaluation of expressions in `if` and `while` was also available in MATLAB 6.0, but was not documented.

New MATLAB Search String Function

`strfind` is a new character array function in MATLAB. It searches for all occurrences of a string pattern within another, longer string. Placement of the two string arguments in the argument list requires that you be specific about which string is the character pattern to search for and which is the string in which to search. This allows you more control over how the search is performed compared with the MATLAB `findstr` function, particularly when executing searches within a loop.

New File I/O Functions for Scientific Data Formats

There are six new MATLAB 6.1 functions that enable you to retrieve information and data from Common Data Format (CDF), Flexible Image Transport System (FITS), and Hierarchical Data Format (HDF) files.

Function	Purpose
`cdfinfo`	Return information about a CDF file
`cdfread`	Read data from a CDF file
`fitsinfo`	Return information about a FITS file
`fitsread`	Read data from a CDF file
`hdfinfo`	Return information about an HDF or HDF-EOS file
`hdfread`	Read data from an HDF or HDF-EOS file

New Audio Functions

MATLAB 6.1 includes two new audio functions for 32-bit Windows platforms only.

Function	Purpose
`audioplayer`	Create an audio object to play audio data
`audiorecorder`	Create an audio object to record audio data

Date Conversion Changes

The `datenum` and `datestr` functions can now accept a date vector, as defined by `datevec`, as an input argument. For example, `datestr(clock)` returns the current date and time as string such as `27-Apr-2001 15:58:41`.

Graphics Features

Transparent Legend

You can now make the legend box transparent, enabling you to see the plotted data behind the legend. See `legend` for more information.

New Ghostscript Drivers

The following new Ghostscript drivers are available with MATLAB by using the device switch shown below.

Printer Driver	Device Switch
Canon Color BubbleJet BJC-800	`-dbjc800`
HP LaserJet 4.5L and 5P	`-dljet4`
HP LaserJet 5 and 6	`-dpxlmono`

New Ghostscript Output Filters for Exporting

The following new Ghostscript output filters are available with MATLAB by using the option switch shown below.

File Format	Option Switch
BMP Monochrome BMP	`-dbmpmono`
PDF Color file Format	`-dpdf`

Higher Resolution Metafiles

You can now set the resolution of a Windows Enhanced Metafile copied from a MATLAB figure window with the `print -dmeta` command. Set the resolution using the `-d` option of the `print` command. For example, to copy a figure to a metafile having a resolution of `200` dpi, use

```
print -dmeta -r200
```

MATLAB uses the screen resolution as the default.

Default PaperType and PaperUnits Set For International Users

The `matlabrc.m` startup file now sets the default `PaperType` and `PaperUnits` properties based on ISO Country Codes. These default to `'a4'` and `'centimeters'` respectively for users in countries that normally default to these settings. Other countries still default to `'usletter'` and `'inches'`.

The same values are used for default Simulink `PaperType` and `PaperUnits` properties in the `matlabrc.m` startup file.

You can still set default `PaperType` or `PaperUnits` values yourself by adding the following to `startup.m`.

```
set(0, 'DefaultFigurePaperType', 'a4')
set(0, 'DefaultFigurePaperUnits', 'centimeters')
```

OpenGL Renderer Feature — Microsoft Windows

If you do not want to use hardware OpenGL, but do want to use object transparency, you can issue the following command.

```
feature('UseGenericOpenGL',1)
```

This command forces MATLAB to use generic OpenGL on Microsoft Windows platforms. Generic OpenGL is useful if your hardware version of OpenGL does not function correctly and you want to use image, patch, or surface transparency, which requires the OpenGL renderer.

To reenable hardware OpenGL, use the command

```
feature('UseGenericOpenGL',0)
```

Note that the default setting is to use hardware OpenGL

To query the current state of the generic OpenGL feature, use the command

```
feature('UseGenericOpenGL')
```

See the `opengl` reference page for additional information.

External Interfaces/API Features

Concatenation of Java Arrays

In MATLAB 6.1, you can concatenate arrays of Java objects that have unlike dimensions. The following example concatenates a 2-by-3 array of `java.lang.Integer` with a 4-by-3 array of the same class.

```
A =
java.lang.Integer[][]:
   [ 1]    [ 2]    [ 3]
   [ 4]    [ 5]    [ 6]
   [17]    [18]    [19]
   [20]    [21]    [22]

B =
java.lang.Integer[][]:
   [11]    [12]    [13]
   [14]    [15]    [16]
```

The vertical concatenation `[A;B]` is simple since both arrays have the same number of columns. The horizontal concatenation `[A B]` merges the two arrays into an irregularly shaped array having six columns in the first and second rows and three columns in the third and fourth rows.

```
C = [A;B]
C =
java.lang.Integer[][]:
    [ 1]    [ 2]    [ 3]
    [ 4]    [ 5]    [ 6]
    [11]    [12]    [13]
    [14]    [15]    [16]
    [17]    [18]    [19]
    [20]    [21]    [22]
```

```
C = [A B]
C =
java.lang.Integer[][]:
   [6 element array]
   [6 element array]
   [3 element array]
   [3 element array]
```

Note "Concatenation of Java Objects" on page 3-27 discusses changes to how Java objects are concatenated.

New Fortran MX, MEX, MAT, and ENG Functions

The following functions have been added to the Fortran MX, MEX, MAT, and Engine external interface. Most of these functions already exist in the MATLAB C language API.

Table 3-1: New Fortran MX Functions

mxAddField	mxCalcSingleSubscript
mxClassIDFromClassName	mxClearLogical
mxCopyComplex8ToPtr	mxCopyInteger1ToPtr
mxCopyInteger2ToPtr	mxCopyPtrToComplex8
mxCopyPtrToInteger1	mxCopyPtrToInteger2
mxCopyPtrToReal4	mxCopyReal4ToPtr
mxCreateCellArray	mxCreateCellMatrix
mxCreateCharArray	mxCreateCharMatrixFromStrings
mxCreateDoubleMatrix	mxCreateNumericArray
mxCreateNumericMatrix	mxCreateScalarDouble
mxCreateStructArray	mxCreateStructMatrix
mxDestroyArray	mxDuplicateArray
mxGetCell	mxGetClassID
mxGetClassName	mxGetData
mxGetDimensions	mxGetElementSize
mxGetEps	mxGetField
mxGetFieldByNumber	mxGetFieldNameByNumber
mxGetFieldNumber	mxGetImagData
mxGetInf	mxGetNaN
mxGetNumberOfDimensions	mxGetNumberOfElements
mxGetNumberOfFields	mxIsCell

Table 3-1: New Fortran MX Functions (Continued)

mxIsChar	mxIsClass
mxIsEmpty	mxIsFinite
mxIsFromGlobalWS	mxIsInf
mxIsInt8	mxIsInt16
mxIsInt32	mxIsLogical
mxIsNaN	mxIsSingle
mxIsStruct	mxIsUint8
mxIsUint16	mxIsUint32
mxMalloc	mxRealloc
mxRemoveField	mxSetCell
mxSetData	mxSetDimensions
mxSetField	mxSetFieldByNumber
mxSetImagData	mxSetLogical

Table 3-2: New Fortran MEX Functions

mexFunctionName	mexGetArray
mexGetArrayPtr	mexIsGlobal
mexIsLocked	mexLock
mexMakeArrayPersistant	mexMakeMemoryPersistant
mexPutArray	mexUnlock
mexWarnMsgTxt	

Table 3-3: New Fortran MAT Functions

matDeleteArray	matGetArray
matGetArrayHeader	matGetNextArray

Table 3-3: New Fortran MAT Functions (Continued)

matGetNextArrayHeader	matPutArray
matPutArrayAsGlobal	

Table 3-4: New Fortran Engine Functions

engGetArray	engPutArray

Property Added to ActiveX and Engine Interfaces

For ActiveX automation server applications and MATLAB Engine applications running on Windows, you can control whether the application windows appear on the Windows desktop with a new property called `Visible`.

When `Visible` is set, the ActiveX application or engine server window is visible on the desktop, thus enabling user interaction with the server. This is the default. When `Visible` is cleared, the application or engine window is removed from the desktop.

ActiveX. This example disables the visibility of an ActiveX automation server application by setting `h.visible` to `0`. It checks the visibility setting in line 3 by examining `h.visible`.

```
h = actxserver('Matlab.Application');
h.visible = 0;

h.visible
ans =
     0
```

MATLAB Engine. For a MATLAB engine session, use the `engSetVisible` and `engGetVisible` functions that are new in MATLAB 6.1. Line 4, below, disables the visibility of the MATLAB engine window using `engSetVisible` with an argument of `0`. Line 5 checks this setting with `engGetVisible`.

```
Engine *ep;
bool vis;
ep = engOpen(NULL);
engSetVisible(ep, 0);
engGetVisible(ep, &vis);
```

Serial I/O

The MATLAB serial port interface provides direct access to peripheral devices such as modems, printers, and scientific instruments that you connect to your computer's serial port. This interface is established through a serial port object, which you create with the `serial` function.

Freeing the Serial Port on Windows Platforms. The serial port object uses the `javax.comm` package to communicate with the serial port. However, due to a memory leak in `javax.comm`, the serial port object is not released from memory. You can use the `freeserial` function to release the MATLAB hold on the serial port.

`freeserial` is necessary only on Windows platforms. You should use `freeserial` only if you need to connect to the serial port from another application after a serial port object has been connected to that port, and you do not want to exit MATLAB.

Events, Callbacks, and Function Handles. Action properties and action functions are now referred to as callback properties and callback functions. This new terminology is reflected in new names for the associated properties and functions. The general rule for the name changes is to change "Action" to "Fcn" for properties, and "action" to "callback" for functions. For example, `TimerAction` has been renamed `TimerFcn`, and `instraction` has been renamed `instrcallback`.

Additionally, if you want to automatically pass the object and event information to the callback function, then you must specify the function as either a function handle or as a cell array. Note that you can also specify the callback function as a string. In this case, the callback is evaluated in the MATLAB workspace and no requirements are made on the function's input arguments.

Enhancements to Existing Properties.

- `Terminator` Property – You can configure `Terminator` to a decimal value ranging from 0 to 127, to the equivalent ASCII character, to `CR/LF` or `LF/CR`, or to empty ('').
- Timer events – Some timer events may not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small. The minimum `TimerPeriod` value is now 0.01 second.

Creating Graphical User Interfaces — GUIDE

This section lists the changes made to GUIDE for Release 12.1:

- The Layout Editor **Edit** menu has **Undo** and **Redo** items. You can undo or redo layout actions and property settings (with the exception of the figure `FileName` property).
- The **Application Option** dialog supports a new option for **Command-line accessibility** – Callback. This option is now the default.
- The Layout Editor displays the layout grid in the current figure color.
- The Layout Editor context menus have been reorganized.
- The Menu Editor enables you to rearrange the order of menu items.
- The Menu Editor adds callback function stubs to the application M-file.

See Creating Graphical User Interfaces in the MATLAB documentation for more information.

Major Bug Fixes

MATLAB 6.1 includes several bug fixes made since MATLAB 6.0. This section describes the particularly important bug fixes.

Also, MATLAB 6.1 includes several important MATLAB 6.0 bug fixes.

Development Environment

Help Browser Supports Mouse Wheel

For Windows platforms, the wheel on your mouse will now work in the Help browser.

UNIX Help Browser Search Results Now Highlighted

On UNIX systems, when you perform a full text search using the Help browser, the search terms are highlighted when you view a page.

UNIX Paste Problems Fixed

On some UNIX systems, pasting after a cut or copy would sometimes cause the system to hang. That problem has been fixed. However, due to issues with UNIX itself, the paste does not always work and you might have to do it again.

Mathematics

Memory Leak Fixed in Matrix Multiply

Under certain conditions, matrix multiply (which includes matrix-vector multiply, vector-matrix multiply, and even vector inner products) leaked memory. For example, on a Pentium III under Linux or Windows, any vector inner product of length greater than 15,000 leaked memory. This was observed by MATLAB increasing its use of system resources that were never returned. MATLAB 6.1 uses new ATLAS BLAS libraries that no longer leak memory.

Improved Convergence for eigs(A,k,'sm') and eigs(A,k,0)

In MATLAB 6.0, `eigs` was reimplemented to use the ARPACK library of routines. Unfortunately, the smallest magnitude case, `sigma = 'sm'` and `sigma = 0`, chose the wrong algorithm. For MATLAB 6.1, the correct ARPACK algorithm is used and convergence is much quicker.

This bug fix introduces a backwards incompatibility. When `A` is a function `Afun` and `sigma = 'sm'`, `Afun` must now return `Y = A\x`. Prior to MATLAB 6.1, `eigs` required `Afun` to return `y = A*x` for this case.

quad Sampling Improved

In MATLAB 6.0, `quad('cos(4*n*x)',-pi,pi)` returned `2*pi` instead of `0`. When `quad` initially sampled the function, it incorrectly assumed the function is the constant 1 over the interval `[-pi,pi]` and so returned `2*pi` early. It now samples more carefully and returns `0`.

griddata3 Inner Matrix Error Message

In MATLAB 6.0, an internal error sometimes caused `griddata3` to display the error message, `Inner matrix dimensions must agree`. This error has been corrected.

Improved Handling of Degenerate Triangulation

In MATLAB 6.0, there were sometimes problems associated with degenerate triangulation. For example, `convhull` could produce a convex hull that did not cover all the original data. MATLAB 6.1 corrects this problem by replacing the utility function `delaunayc` with Qhull.

Error Message Display for Qhull-Related Functions

In MATLAB 6.0, Qhull-related functions (e.g., `delaunayn`) displayed error messages in standard error. For UNIX platforms, standard error is different from the command window. For MATLAB 6.1, error messages are displayed in the command window.

histc Computes First Two Bins Correctly

Prior to MATLAB 6.0, `histc` produced the wrong results for the first two bins for cases with extremely nonuniform bin edges. This problem was corrected in MATLAB 6.0.

```
Processor has been on-line since 04/20/2001 14:09:31
The alpha EV4.5 (21064) processor operates at 233 MHz,
and has an alpha internal floating-point processor.
```

The number in parentheses on the third line, in above example (21064), is the number you are interested in.

GLNX86. Enter the following command

```
cat /proc/cpuinfo
```

and look for the following fields in the output (values may vary from the example below)

```
vendor_id  :GenuineIntel
cpu family :6
model      :8
model name :Pentium III (Coppermine)
stepping   :1
```

Match up this information with the table in `<MATLAB>\bin\glnx86\blas.spec`.

Note Some versions of glibc 2.1.x have problems with environment variables (and the ability to reliably query them) from within shared library `init` functions. To take advantage of the `BLAS_VERSION` feature, you may need to upgrade your machine to glibc 2.2.

HP700. Start with the **System Administration Manager (SAM)** and work your way to the **Processor** tab, as shown below:

System Administrator Manager (SAM) -> **Performance monitors** -> **System properties** -> **Processor** tab

This provides information about the type of processor.

HPUX. MATLAB only supports HPUX running on PA-RISC2.0.

IBM_RS. Contact IBM Technical Support and request the document entitled "Determining CPU Speed in AIX." This is a table of machine types, processor types, and processor speeds.

SGI. Enter the following command

```
sysinfo -a
```

which returns a lot of information. In the first few lines, look for information something like

```
CPU Type is              mips R4400 5.0
```

The information starting with `R` is what you are interested in. MATLAB ships for the `R5000`, `R8000`, `R10000` and `R12000` (default).

SOL2. Enter the following command

```
uname -m
```

which returns either `sun4u` for UltraSPARC or `sun4m` for the older, non-Ultra machines (e.g., Hyper and SuperSPARCs).

WIN32. Start with the **My Computer** icon, and work your way to the **General** tab, as shown below:

My Computer -> **Control Panel** -> **System** -> **General** tab

This should list the family and model number for your computer. On Windows NT and Windows 2000, the same information is on the **Environment** tab, under the System Variable `PROCESSOR_IDENTIFIER`. Match up this information with the table in `<MATLAB>\bin\win32\blas.spec`.

Using Another BLAS

You may also use BLAS from other sources than the ones shipped with MATLAB, provided they are in the correct format. This format is a shared library (as opposed to a static library) that exports all the double-precision (starting with d) and double-precision complex (starting with z) BLAS routines from `dasum` to `zupmtr`. On HP, IBM_RS, and WIN32, the symbols must be exported without trailing underscores, while for ALPHA, GLNX86, SGI, and SOL2, the symbols must be exported with trailing underscores (e.g., `dgemm_`).

If the shared library you provide also includes LAPACK symbols like `dgefa` (or `dgefa_`), then they will override the MATLAB default implementation, which is based on the Fortran LAPACK from Netlib at `http://netlib.org`.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from MATLAB 6.0 (Release 12.0) to MATLAB 6.1 (Release 12.1). This section about upgrading from an earlier release is organized into the following subsections:

For information about upgrading from an earlier version than MATLAB 6.0, see "Upgrading from an Earlier Release" on page 4-42 in the MATLAB 6.0 Release Notes.

Development Environment Issues

subscribe Function No Longer Supported

The `subscribe` function is no longer supported.

Command History, Preferences, and Favorites

If you uninstall Release 12.0, you will lose the Command History, preferences, and Help browser favorites from Release 12.0.

To keep these files for use in Release 12.1, make a copy of them before uninstalling Release 12. To see where the files are located, run `prefdir` in the Command Window. The relevant files are listed below.

Filename	File For
`cwdhistory.m`	Command Window history
`history.m`	Command History
`matlab.prf`	Preferences
`matlab_help.hst`	Help browser favorites

After uninstalling Release 12, put your backup copy of the files in the location returned by `prefdir` so that Release 12.1 can use the files.

Help Browser Favorites

If you use favorites you created for the documentation in the Release 12.0 Help browser, those favorites may point to an incorrect or invalid location in Release 12.1. You will need to delete any invalid favorites and add those favorites again.

Source Control

If you use Microsoft Visual SourceSafe with the MATLAB source control features, you now need to specify the login information for SourceSafe using preferences. Select **File -> Preferences -> General -> Source Control** from the desktop. Specify the **Username**, **Password**, and **Database**.

Mathematics Issues

Finding Smallest Magnitude Eigenvalues

`eigs(A,k,sigma)` and `eigs(A,B,k,sigma)` return `k` eigenvalues based on `sigma`. For `sigma = 'sm'`, `eigs` returns the smallest magnitude eigenvalues.

In MATLAB 6.0, `eigs` was reimplemented to use the ARPACK library of routines. Unfortunately, the smallest magnitude case, `sigma = 'sm'` and `sigma = 0`, chose the wrong algorithm. For MATLAB 6.1, the correct ARPACK algorithm is used and convergence is much quicker.

This bug fix introduces a backwards incompatibility. When `A` is a function `Afun` and `sigma = 'sm'`, `Afun` must now return `Y = A\x`. Prior to MATLAB 6.1, `eigs` required `Afun` to return `y = A*x` for this case.

Possible Changes in Results Returned by Matrix Functions

Starting in MATLAB 6.0 (R12.0), matrix computations are based on LAPACK, a large, multiauthor Fortran subroutine library for numerical linear algebra. While this change has many benefits and matrix functions continue to operate in the same way in MATLAB 6.0, the results returned by matrix functions may differ. Changes in roundoff errors can be seen in most matrix computations. In cases where quantities are not uniquely determined mathematically, results may differ in order and in normalization.

For example:

- Eigenvalues may be returned in a different order.
- Eigenvectors may be normalized differently.
- The signs of columns of orthogonal matrices may differ.
- `rcond` is a better estimate of the reciprocal condition.
- `lu` can now be used to factor rectangular full matrices.

Obsolete Input Arguments

Certain input arguments to these functions have become obsolete. Using these arguments does not result in an error, but they are ignored.

Function	Description
`delaunay`	Now ignores the third argument `fuzz`, which specified a value for the `fuzz` standard deviation. Now ignores the third argument `'sorted'`. This argument indicated to `delaunay` that the given points `x` and `y` were sorted, and that duplicate points had been eliminated.
`convhull`	Now ignores the third argument `TRI`, which provided triangulation data previously computed using `delaunay`.

Obsolete Functions

The following MATLAB function has become obsolete. For backwards compatibility, it has not been removed from the language at this time. However, this function may be removed in a future release, and you are encouraged to discontinue its use, or use the function that replaces it

Function	Description
`bvpval`	Evaluate the numerical solution of a boundary value problem (BVP). Replace with `deval`, which evaluates the solution of both initial value and boundary value differential equation problems.

Programming and Data Types Issues

Output from Background and Foreground Commands (UNIX)

In Release 12 on UNIX platforms, a background command (i.e., any system command after which you add a `&`), such as

```
! cat startup.m &
```

no longer produces any output. Prior to Release 12, a background command sent output to the command window.

If you need to see the output from a command, either do not make the command a background command (i.e., remove the `&`), or run the background command in a separate xterm. To start another xterm, issue the following command.

```
! xterm &
```

In Release 12, foreground functions (i.e., nonbackground functions) send their output to the diary, if the `diary` function has been issued. The output is also displayed in the command window (prior to Release 12, foreground function output was only displayed in the command window).

matlab_helper Process

To make the `!` and `unix` commands operate more efficiently, in Release 12 MATLAB creates a secondary process, called `matlab_helper`, at startup.

This `matlab_helper` contains those elements of MATLAB necessary to run the `!` and `unix` commands.

Graphics Issue

MATLAB No Longer Supports Terminal Mode

MATLAB no longer runs on nongraphics computer terminals.

External Interfaces/API Issues

Concatenation of Java Objects

When you concatenate Java objects, the class of the resultant object depends on the classes of the input objects, as follows:

- If the input objects are of the same class, MATLAB makes the output object of that class. This was true in Version 6.0 as well.
- If the input objects are of different classes, but all inherit from a common class, MATLAB makes the output object of the common parent class. MATLAB selects the lowest common parent in the Java class hierarchy as the output class. This is new behavior for Version 6.1.

  For example, concatenating objects of classes `java.lang.Integer` and `java.lang.Double` creates a new object of class `java.lang.Number`.
- If the input objects are of different and unrelated classes, then MATLAB makes the output object of the `java.lang.Object` class. This was true in Version 6.0 as well.

Obsolete Fortran MX, MEX, MAT, and ENG Functions

The following Fortran MX, MEX, MAT, and ENG functions are considered to be obsolete as of Version 6.1. Support for these functions may be removed from a future MATLAB release.

Table 3-5: Obsolete Fortran MX Functions

`mxCreateFull`	`mxFreeMatrix`
`mxIsFull`	`mxIsString`

Table 3-6: Obsolete Fortran MEX Functions

mexGetEps	mexGetGlobal
mexGetFull	mexGetInf
mexGetMatrix	mexGetMatrixPtr
mexGetNaN	mexIsFinite
mexIsInf	mexIsNaN
mexPutFull	mexPutMatrix

Table 3-7: Obsolete Fortran MAT Functions

matDeleteMatrix	matGetFull
matGetMatrix	matGetNextMatrix
matGetString	matPutFull
matPutMatrix	matPutString

Table 3-8: Obsolete Fortran Engine Functions

engGetFull	engGetMatrix
engPutFull	engPutMatrix

Known Software and Documentation Problems

This section updates the MATLAB 6.1 documentation set, reflecting known MATLAB 6.1 software and documentation problems. It is organized into the following subsections:

Development Environment Problems

Displaying Results From lookfor Function

When you run the `lookfor` function, press **Ctrl+C** to display the results in the command window.

Many Open Windows Can Cause a Crash or Hang (Windows 98/Me)

On Microsoft Windows 98/Me platforms, if you keep many windows open, MATLAB may crash or hang. For example, if you keep open 12 Stateflow windows, or 25 figure windows, or 50 Simulink windows, you may experience this problem. Note that these numbers are only estimates; the actual number of open windows that may cause this problem depends on the resources currently in use by other components and applications.

Cannot Go To Top Level of UNC Path

For Windows platforms, you cannot use `cd` or any directory tool in the MATLAB desktop (including the Current Directory browser and Set Path dialog box) to access the top level of a UNC path.

Workspace Browser with Many Variables

If there are many variables in the workspace and the Workspace browser is open, you may experience performance problems. If you expect to have more than 1000 variables in the workspace, close the Workspace browser to avoid performance problems.

UNIX Display Problems When UNIX Client and Server Platforms Differ

If you use MATLAB on UNIX and the platform for the server is different than that for the client, there may be problems with the display of graphics on the

client. See the Technical Support Web page for a solution that lists the combinations tested and any known display problems with them.

Sun Solaris 16-Bit Display Not Supported

Sun's Java VM for Solaris does not support 16-bit displays. Therefore you cannot use this configuration with Release 12. Use another display mode instead.

Sun Solaris Arrow Keys Not Working

On some Sun Solaris systems, the arrow keys on the main keyboard are not working properly. Instead, try the arrow keys in the numeric keypad.

Alpha Shortcut Problems When Using Emacs Key Bindings in Editor

On the Alpha platform, if you set the Editor/Debugger preference for key bindings to Emacs, the shortcuts for **Undo** (**Ctrl+_**) and **Copy** (**~+W**) do not work.

Display Problems with Xoftware

If you use Xoftware on a PC to run MATLAB on a UNIX platform, you need to do the following to avoid display problems:

1. Go to the Xoftware **Control Panel**.
2. From the **Options** menu, select **Configuration**.
3. Select the **Window** tab.
4. From the **Options** listing, select **Concurrent Window Manager**.
5. Under **Settings**, select **Off**.
6. Click **OK**.

Documentation Updates

Editor/Debugger Example - Graphic and Information Incorrect

In the printed book *Using MATLAB* (Version 6), on page 7-19, the graph shown is incorrect. For the correct graph, see the same page in the Help browser at

MATLAB -> Using MATLAB -> Development Environment -> Editing and Debugging M-Files -> Debugging M-Files -> Trial Run for Example.

On page 7-29, in "Correcting Problems and Ending Debugging, Completing the Example," step 3 is incorrect. It should instead read "In `collatzplot.m` line 12, change the string `plot_seq` to `seq_length(m)` and save the file."

interp1 Extrapolation of Out-of-Range Values

A new argument enables `interp1` to perform extrapolation for out-of-range values for all methods. It also enables you to specify a scalar to be returned for out-of-range values.

The PDF version of the `interp1` reference page incorrectly states that the default for all methods is for `interp1` to perform extrapolation for out-of-range values. In fact, `interp1` performs extrapolation as the default only for the `'spline'`, `'pchip'`, and `'cubic'` methods. For all other methods, it returns `NaN` for out-of-range values. This behavior is unchanged from Version 5.

The HTML reference page for `interp1` is correct.

4

MATLAB 6.0 Release Notes

New Features

This section introduces the new features and enhancements added in MATLAB 6.0 since MATLAB 5.3 (Release 11.0).

This section about new features is organized into the following subsections:

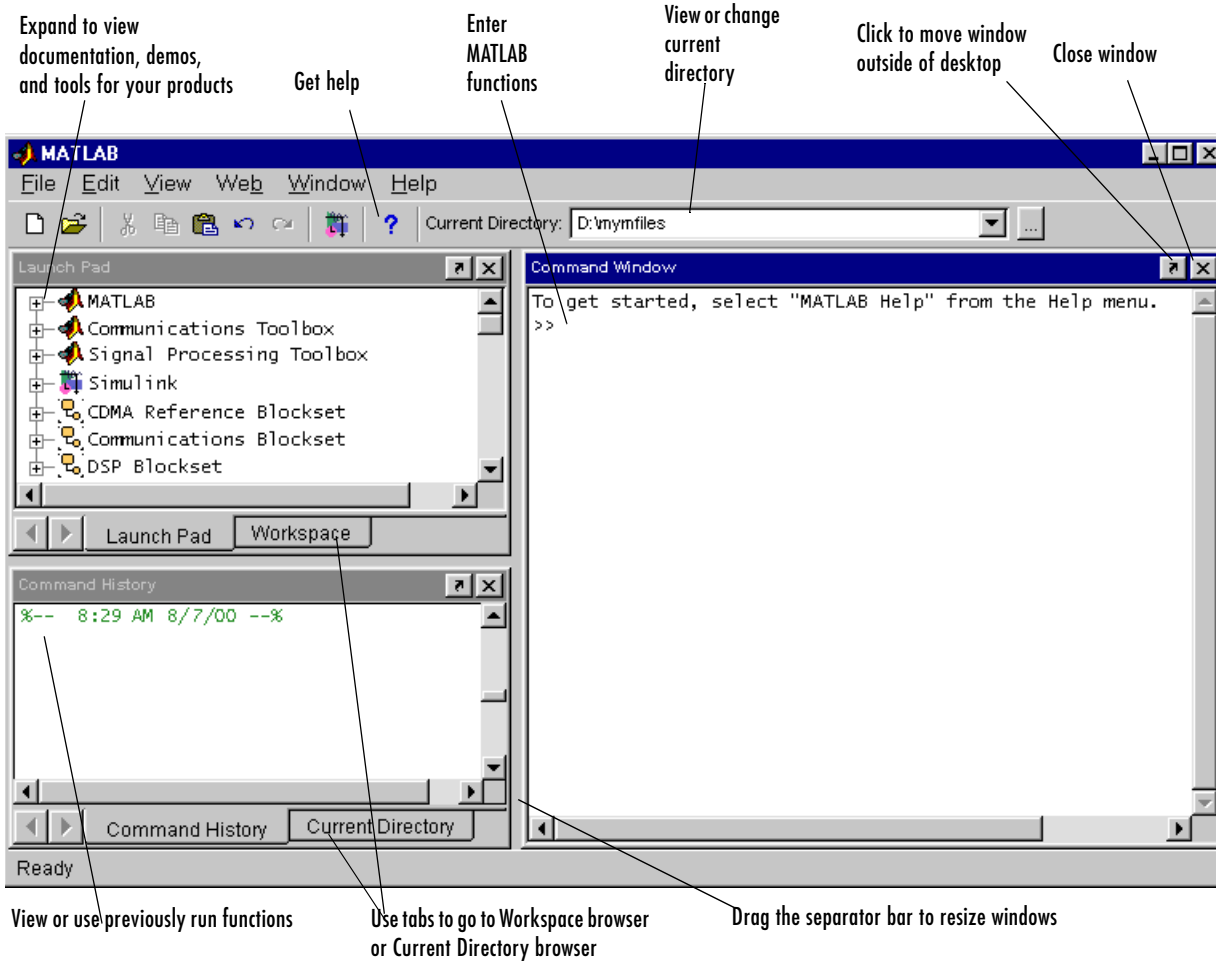
Development Environment Features

This section includes

MATLAB Desktop

MATLAB has a new environment called the MATLAB desktop, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB. Think of the desktop as your MATLAB dashboard. The first time MATLAB starts, the desktop appears as shown in the following

illustration, although your Launch Pad may contain different entries. You can change the way your desktop looks by opening, closing, and moving tools.



If you prefer a command line interface, you can use functions to perform most of the features found in the MATLAB desktop tools.

The following tools are managed by the MATLAB desktop, but not all of them appear by default when you first start.

Tool	Purpose	Major New Features
Command Window	Run MATLAB functions.	• Context menu to evaluate, open, or get help for a selection • Tab completion for function names • Syntax highlighting
Command History	View a log of the functions you entered in the command window, copy them, and execute them.	New tool
Launch Pad	Easily run tools and access documentation for all your MathWorks products. Expand a listing to show the documentation, demos, and tools for that product.	New tool
Help browser	View and search the documentation for the MATLAB full product family, as described in “New Online Help” on page 4-6.	New user interface that replaces the Help Desk
Current Directory browser	View MATLAB files and related files. Perform file operations such as open files, and find and replace strings in a file.	New tool

Tool	Purpose	Major New Features
Workspace browser	View and make changes to the contents of the workspace.	• Interface to the Import Wizard • Graph data for a variable using the context menu
Array Editor	View array contents in a table format and edit the values.	• Change the display format for variables • Allow strings and cell arrays of strings
Editor/ Debugger	Create, edit, and debug M-files (files containing MATLAB functions).	• Show line numbers • Comment or uncomment a selection (multiple lines) • Change colors used for syntax highlighting • Search through multiple files at once for a specified phrase • Option for MATLAB to start a session and open files that were open at shutdown of the previous session • Option to view datatips in edit mode • Breakpoints are maintained when a file is saved.

To see how many of these new features work, select **Demos** from the **Help** menu in the desktop. The playback demos run in your system's Web browser and illustrate the main features of the new interfaces and tools. The playback demos run faster on Windows platforms than they do on UNIX.

Other MATLAB tools are not managed by the desktop, such as figure windows, toolbox graphical user interfaces, and the following development environment tools.

Tool	Purpose	Major New Features
Set Path dialog box	View and change the MATLAB search path.	A modified version of the Path Browser user interface
Import Wizard	Load binary or text data into the MATLAB workspace.	Graphical user interface to the MATLAB import functions. See “Import Wizard” on page 4-8 for more information.
M-File Profiler	Measure where an M-file is spending its time to help you make speed improvements.	Now supports scripts
Source Control Interface	Access your source control system from within MATLAB, Simulink, and Stateflow.	New tool
Notebook	Access the MATLAB numeric computation and visualization software from within a word processing environment (Microsoft Word).	Supports Word 2000

New Online Help

Release 12 provides almost all the documentation in online form in HTML (a few products' online documentation is in PDF form only). The online documentation is at least as up-to-date as any printed documentation shipped with the product, and in several cases is more up-to-date.

New Help Browser. Release 12 includes its own Help browser, which allows you to access the online documentation similarly to how you do with Microsoft's HTML Help and Sun's Java Help interfaces. However, the Help browser has been customized to work even more effectively with the whole MATLAB product family. The access methods (via tabs) include

- An expandable/collapsible table of contents, organized by products
- An index
- A search facility, including a full-text search, as well as searches for word(s) in documentation section titles, function names, or even the Technical Support online knowledge base (via the Web)

Other features include saving favorites (bookmarks) and being able to execute code examples in the online documentation by highlighting the text and using a right-click context menu.

You can print out copies of the documentation by accessing PDF versions of the documentation. For Windows platforms, PDF files are on the Documentation CD.

See "Getting Help" in the MATLAB documentation for complete instructions.

Context-Sensitive Help. For several products, you can access context-sensitive help via Help menus, Help buttons, or from a right-click context menu.

Toolbox Path Cache Reduces MATLAB Startup Time

If you run MATLAB from a network server, you can significantly reduce your startup time by using the new toolbox path cache feature. The toolbox path cache stores path information on all toolbox directories under the MATLAB root directory. During startup, MATLAB obtains this information from the cache rather than by reading it from the remote file system.

The toolbox path cache is used only during the startup of your MATLAB session. It is especially useful if you define your MATLAB path to include many toolbox directories. It takes considerable time to acquire all of this information by scanning directories in the remote file system. Reading it from a pregenerated cache however, is significantly faster. If you have a short toolbox path, there is less benefit to using the cache, but it does still provide a time savings.

When you first install MATLAB, the toolbox path cache must be generated by the system administrator and enabled on those systems for which it is needed. The MATLAB **Preferences** dialog box has a new **Toolbox Caching** panel that assists you in generating and enabling the cache.

See "Reducing Startup Time with Toolbox Path Caching" in the "Development Environment" of the MATLAB documentation for more information.

Import Wizard

The easiest way to import ASCII text data or binary data into the workspace is to use the new MATLAB Import Wizard. To use the Import Wizard, follow these steps:

1. Start the Import Wizard, by choosing the **Import Data** option on the MATLAB command window **File** menu. The Import Wizard displays a file selection dialog box. Specify the file that contains the data you want to import.

2. The Import Wizard opens the file, processes the data in the file, and displays a preview of the variable (or variables) it has created from the data in the file. If the file contains multiple variables, you can select the variables you want to import. Click **Finish** to import the data into the workspace.

The Import Wizard can process many types of data formats automatically, such as images, sound files, and spreadsheets. The Import Wizard can also process text data files that use commas, spaces, tabs or semicolons as delimiters. (The delimiter, also known as a column-separator, is the character used to separate the individual data items in text data file.) The Import Wizard can process other text files if you specify the delimiter used in the file.

The following table lists the types of data you can import using the Import Wizard.

Data Types	File Extension
ASCII text data	`.txt`, `.dat`, `.dlm`, and others
Audio Video Interleaved (AVI) format	`.avi`
CompuServe Graphics Interchange format	`.gif`

Data Types	File Extension (Continued)
Cursor format	`.cur`
HDF raster image	`.hdf`
Icon	`.ico`
JPEG	`.jpg`, `.jpeg`
MATLAB MAT-file	`.mat`
Portable Network Graphics	`.png`
Sound files	`.wav`, `.au`, `.snd`
Spreadsheet	`.csv`, `.xls`, `.wk1`
Zsoft Paintbrush	`.pcx`

Development Environment Functions

This section lists the new and changed development environment functions.

New Development Environment Functions. The functions listed in the following table are new in MATLAB 6.0.

Function	Description
`checkin`	Check files into your source control system from MATLAB, Simulink, and Stateflow.
`checkout`	Check files out of your source control system into MATLAB, Simulink, and Stateflow.
`cmopts`	Get the name of the source control system being used with MATLAB.
`customverctrl`	Integrate a version control system not supported with MATLAB.

Function	Description (Continued)
`filebrowser`	Display the Current Directory browser, a tool for viewing files in the current directory and performing file operations.
`helpbrowser`	Display the MATLAB Help browser, which provides access to extensive online help.
`undocheckout`	Undo the previous checkout from the source control system.

Development Environment Functions That Changed. The functions listed in the following table have been changed since MATLAB 5.2 (Release 11).

Function	Description of Change
`dbstop`	The function `dbstop if error` no longer stops execution on errors detected within a `try...catch` block. MATLAB does not enter debug mode under these circumstances. Use the new form of the function, `dbstop if all error` to stop execution and enter debug mode on all types of errors, including those detected within a `try...catch`.
`doc`	Displays documentation in the Help browser instead of in the Help Desk. If an HTML reference page for a function does not exist, it displays M-file help in the Help browser.
`docopt`	Is now only used for: • The `web` function, if the `-browser` option is used for UNIX platforms • The IBM and HP platforms

Function	Description of Change (Continued)
`helpdesk`	Displays the Help browser instead of the Help Desk. In a future release, the `helpdesk` function will be phased out.
`helpwin`	Function listings and descriptions appear in the Help browser instead of in a specialized window.
`pathtool`	Opens the new **Set Path** dialog box instead of the Path Browser.
`version`	Now has a `-java` flag, which displays the version of Java used by MATLAB.
`web`	By default, displays the specified `URL` in the Help browser. Now includes the `-browser` flag, which displays the specified `URL` in your system's default Web browser.

Mathematics Features

The following mathematics features have been added or enhanced in MATLAB 6.0:

- Matrix computations
- N-dimensional Delaunay-type functionality
- Differential equations solvers
- Sparse matrices
- Fast Fourier transforms
- Quadrature
- Function functions
- Basic Fitting interface
- Data Statistics interface

These features are described below. At the end of this section are tables that summarize changes to the MATLAB math functions:

- New functions
- Functions with new or changed capabilities

See "Upgrading from an Earlier Release" on page 4-42 for information about obsolete functions.

Matrix Math in MATLAB 6.0

Matrix computations in MATLAB 6.0 are based on LAPACK, a large, multiauthor Fortran subroutine library for numerical linear algebra. LAPACK extends the MATLAB matrix computation capabilities and increases its speed on larger problems. It offers MATLAB a larger class of algorithms from which to choose based on the properties of the matrix arguments. Optimized Basic Linear Algebra Subroutines (BLAS), on all MATLAB platforms, speeds up matrix multiplication and the LAPACK routines themselves. Optimized BLAS is provided by Automatically Tuned Linear Algebra Software (ATLAS).

The LAPACK Users' Guide, Third Edition, is available online at `http://www.netlib.org/lapack/lug/lapack_lug.html`.

Differing Results. Matrix functions in earlier versions of MATLAB continue to operate in the same way in MATLAB 6.0, but the results they return may differ. Changes in roundoff errors can be seen in most matrix computations. In cases where quantities are not uniquely determined mathematically, results may differ in order and in normalization.

For example:

- Eigenvalues may be returned in a different order.
- Eigenvectors may be normalized differently.
- The signs of columns of orthogonal matrices may differ.
- `rcond` is a better estimate of the reciprocal condition.
- `lu` can now be used to factor rectangular full matrices.

Increased Number of Eigenvalue Algorithms. With MATLAB 6.0, there are now eight different eigenvalue algorithms, determined by

- `eig(A)` or `eig(A,B)`
- Real or complex matrices
- Symmetric/Hermitian matrices and `B`, if any, positive definite
- Whether eigenvectors are requested or not

For the symmetric and Hermitian problems, the eigenvalues are real, sorted in increasing order and the eigenvectors are normalized so that

```
V'*V or V'*B*V = I
```

New Decompositions for Real Matrices. The QZ algorithm returns a newly available real decomposition for real matrices. If `A` and `B` are real and not symmetric,

```
[AA,BB,...] = qz(A,B,'real')
```

returns a real triangular matrix `BB` and a real quasitriangular matrix `AA` with 2-by-2 diagonal blocks corresponding to pairs of complex conjugate eigenvalues. Earlier versions of MATLAB produced complex triangular `AA` and `BB` if there were any complex eigenvalues. You can continue to obtain this behavior with either

```
[AA,BB,...] = qz(A,B)
```

or

```
[AA,BB,...] = qz(A,B,'complex')
```

A similar option for the Schur decomposition of a real matrix

```
T = schur(A,'complex')
```

produces a complex decomposition if `A` has any complex eigenvalues.

The flops Function. The incorporation of LAPACK makes it impractical to count floating-point operations. As a result, the `flops` function is inoperative in MATLAB 6.0. It will be discontinued completely in some future version.

With modern computer architectures, floating-point operation counts are no longer a good predictor of execution time. Counts of memory references and cache usage patterns have become more important.

N-Dimensional Delaunay-Type Functionality

New Qhull-based functions extend Delaunay-type functionality:

- `delaunayn` and `delaunay3` for N-dimensional Delaunay tessellation
- `convhulln` N-dimensional convex hull
- `voronoin` for N-dimensional Voronoi diagrams
- `griddatan` and `griddata3` for data gridding and hyper-surface fitting

Differential Equation Solvers

New differential equation solvers expand the capabilties of MATLAB:

- `bvp4c` solves two-point boundary value problems for ODEs by collocation. Supporting functions enable you to set options that affect problem solution, form an initial guess, and evaluate the numerical solution obtained with `bvp4c`.
- `pdepe` solves initial-boundary value problems for parabolic-elliptic PDEs in 1-D. A supporting function enables you to evaluate the numerical solution obtained with `pdepe`.

The ODE solvers now take advantage of function handles and can solve problems without using ODE files. The new syntax is

```
solver(@odefun,tspan,y0,options,p1,p2,...)
```

where `@odefun`, `tspan` and `y0` are required arguments. See the ODE solver and `odeset` reference pages for details.

MATLAB 6.0 supports use of an ODE file for backwards compatibility, but new functionality is available only with the new syntax.

Sparse Matrix Computations

New and upgraded routines provide new capabilities and speed up computations:

- `eigs` and `svds` now use the Fortran library ARPACK.
- New routines, `symmlq`, `minres` and `lsqr`, iteratively solve symmetric indefinite systems and least squares problems.
- New routines, `colamd` and `symamd`, provide approximate minimum degree permutations to help reduce the fill-in of their sparse factors.
- `condest` can now produce more accurate condition estimates.

Fast Fourier Transforms

Fast Fourier transform (FFT) functions now rely on the MIT FFTW library. This results in faster performance for composite, prime, and large prime factor array lengths.

Quadrature

New quadrature algorithms in `quad` and the new function `quadl` are faster, more accurate and more robust in that they handle singularities better. `quadl` replaces the now obsolete `quad8` function.

Interpolation

A new one-dimensional interpolation function, `pchip`, based on piecewise cubic Hermite interpolating polynomials, preserves the shape and monotonicity of the underlying data.

Function Functions

All function functions are now capable of accepting function handles as arguments. Most also accept additional parameters, which they pass to the function that you pass in as an argument.

For information about function handles, see the `function_handle` (`@`), `func2str`, and `str2func` reference pages, and the Function Handles section of "Programming and Data Types" in the MATLAB documentation.

The Basic Fitting Interface

MATLAB supports curve fitting through the Basic Fitting interface. Using this interface, you can quickly perform many curve fitting tasks within the same easy-to-use environment. The interface is designed so that you can

- Fit data using a spline interpolant, a hermite interpolant, or a polynomial up to degree 10.
- Plot multiple fits simultaneously for a given data set.
- Plot the fit residuals.
- Examine the numerical results of a fit.
- Evaluate (interpolate or extrapolate) a fit.
- Annotate the plot with the numerical fit results and the norm of residuals.
- Save the fit and evaluated results to the MATLAB workspace.

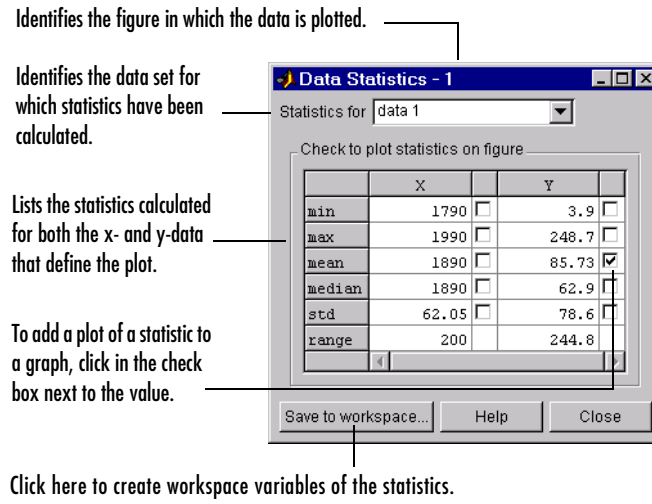
Depending on your specific curve fitting application, you can use the Basic Fitting interface, the command line functionality such as `polyfit` and `polyval`, or both.

The Data Statistics Interface

MATLAB has a new visual interface that:

- Calculates basic statistics about the central tendency and variability of data plotted in a graph
- Lets you save the statistics to the workspace
- Lets you plot any of the statistics in a graph

When you select **Data Statistics** from the MATLAB figure window **Tools** menu, MATLAB calculates the statistics for each data set plotted in the graph and displays the results in a **Data Statistics** dialog box. To plot a statistic in a graph, click in the check box next to its value. To save a set of statistics as a workspace variable, click on the **Save to workspace...** button. The Data Statistics tool saves the statistics as a structure. The following figure shows the components of this dialog box.



Math Function Summary Tables

This section summarizes

- New math functions
- Functions with new or changed capabilities

For more information on these functions, see their respective reference pages or type

```
help function
```

in the MATLAB command window, where `function` is the name of the function about which you want to obtain more information.

Note See “Upgrading from an Earlier Release” on page 4-42 for information about obsolete functions.

New Math Functions

Function	Purpose
`bvp4c`	Solve two-point boundary value problems (BVPs) for ODEs by collocation
`bvpget`	Extract an option from the BVP options structure
`bvpinit`	Form the initial guess for `bvp4c`
`bvpset`	Create/alter BVP options structure
`bvpval`	Evaluate the solution computed by `bvp4c`
`colamd`	Column approximate minimum degree permutation
`convhulln`	N-dimensional convex hull
`delaunay3`	Three-dimensional Delaunay tessellation
`delaunayn`	N-dimensional Delaunay tessellation
`dsearchn`	N-dimensional nearest point search
`griddata3`	Data gridding and hyper-surface fitting for 3-D data
`griddatan`	Data gridding and hyper-surface fitting (dimension >= 2)
`lsqr`	LSQR implementation of Conjugate Gradients on the Normal Equations
`minres`	Solve a system of equations using Minimum Residual Method
`pchip`	Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) – preserves monotonicity and the shape of the data. `pchip` is used by `interp1(x,y,xi,'cubic')`
`pdepe`	Solve initial-boundary value problems for parabolic-elliptic partial differential equations (PDEs) in one dimension
`pdeval`	Evaluate by interpolation the solution computed by `pdepe`

New Math Functions (Continued)

Function	Purpose
`quadl`	Numerically evaluate an integral using adaptive Lobatto quadrature
`symamd`	Symmetric approximate minimum degree permutation
`symmlq`	Solve a system of equations using symmetric LQ method
`voronoin`	Compute N-dimensional Voronoi diagram

Math Functions with New or Changed Capabilities

Function	Purpose
`condest`	`[c,v] = condest(A,t)` specifies a new argument, `t`, a positive integer equal to the number of columns in an underlying iteration matrix. Increasing the number of columns usually gives a better condition estimate but increases the cost. The default is `t = 2`, which almost always gives an estimate correct to within a factor 2.

`dblquad`	`dblquad` now accepts extra arguments `p1,p2,...` which it passes to `fun`. For example,
	`dblquad(fun,xmin,xmax,ymin,ymax,tol,...`
	`@quadl,p1,p2,...)`

`eig`	For symmetric `A` and symmetric positive definite `B`, `eig(A,B,'chol')` computes the generalized eigenvalues of `A` and `B` using the Cholesky factorization of `B`. `'chol'` is the default. `eig(A,B,'qz')` ignores the symmetry, if any, and uses the QZ algorithm.

Math Functions with New or Changed Capabilities (Continued)

Function	Purpose
`eigs`	Now provides an interface to a subset of the ARPACK capabilities. See the `eigs` reference page for information about the expanded syntax, and the `sigma` and `options` arguments. The MATLAB 5 arbitrary ordering of the inputs `B`, `k`, `sigma`, and `options` is no longer allowed.
`fftshift`, `ifftshift`	`fftshift(X,dim)` and `ifftshift(X,dim)` can now apply the shift operation along the dimension specified by `dim`.
`fminbnd`, `fminsearch`, `fzero`, `lsqnonneg`	A new `Display` options parameter value, `'notify'`, displays output only if the function does not converge. For these functions, `'notify'` is the default.
`fzero`	MATLAB Version 5 changed the calling sequence for `fzero`. See the instructions for converting your code in "Function Functions" in the MATLAB documentation.
`gallery`	Two new options produce these test matrices: `'randcolu'` – Random matrix with normalized columns and specified singular values `'randcorr'` – Random correlation matrix with specified eigenvalues
`interp1`	`interp1(x,y,xi,'cubic')` and `interp1(x,y,xi,'pchip')` use `pchip` to perform the interpolation. A new flag `'v5cubic'` provides the cubic interpolation used in MATLAB 5. The default method is `'linear'`. `interp1(x,y,xi,method,'extrap')` uses the specified method to extrapolate any element of `xi` that is outside the interval spanned by `x`. `interp1(x,y,xi,method,extrapval)` returns the scalar `extrapval` for out of range values.

Math Functions with New or Changed Capabilities (Continued)

Function	Purpose
`lu`	`lu(X)` can now be used to factor rectangular matrices.
`ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`	The ODE solvers can now solve problems without the use of an ODE file. Problem components are passed to the solvers directly as arguments, or provided using parameters in an options structure. See the ODE solvers and `odeset` reference pages for details. MATLAB 6.0 supports use of an ODE file for backwards compatibility, but new functionality is available only in the new syntax.
`polyval`, `polyfit`	An optional output argument for `polyfit` and an optional input argument to `polyval` provide for centering and scaling, that is, subtracting the mean and normalizing the standard deviation of the independent variable.
`quad`	`quad(fun,a,b)` uses a new default tolerance, 10^{-6}. Because of the use of new quadrature algorithms, your results (`q`) and the number of function evaluations (`trace(1) = fcnt`) may differ from MATLAB 5. The new algorithm provides more accurate results and generally result in improved performance.
`qz`	For real `A` and `B`, `[AA,BB,Q,Z,V] = qz(A,B,'real')` produces a real decomposition with a quasitriangular `AA`. `[AA,BB,Q,Z,V] = qz(A,B,'complex')` produces a possibly complex decomposition with a triangular `AA`.
`schur`	For real `X`, `schur(X,'real')` has the real eigenvalues on the diagonal and the complex eigenvalues in 2-by-2 blocks on the diagonal. `schur(X,'complex')` is triangular and is complex if `X` has complex eigenvalues.

Math Functions with New or Changed Capabilities (Continued)

Function	Purpose
`sort`	`sort(S)` now works on other data types, for example `int32`, and has been rewritten to be faster on doubles. `sort(S,dim)` can now sort the elements of both full and sparse matrices along the dimension specified by `dim`.
`sqrtm`	`[X, alpha, condest] = sqrtm(A)` returns a stability factor `alpha` and an estimate `condest` of the matrix square root condition number of `X`.
`std`	`std([])` no longer returns empty. It now returns `NaN`, and a message, `Warning: Divide by zero`.

Programming and Data Types Features

MATLAB Interface to Java

MATLAB 6.0 provides an interface to Java that enables you to create objects from Java classes and call methods on those objects. You can use existing Java classes or create your own. See “MATLAB Interface to Java” on page 4-33 for a summary of this feature. See “Calling Java from MATLAB” under “External Interfaces/API” in the online help, for full documentation.

Function Handles

The MATLAB language has a new data type called the function handle. You can create a handle to any MATLAB function and then use that handle as a means of referencing the function. A function handle is typically passed in an argument list to other functions, which can then execute, or *evaluate*, the function using the handle.

A MATLAB function handle is more than just a reference to a function. It often represents a collection of function methods, overloaded to handle different argument types. When you create a handle to a function, MATLAB takes a snapshot of all built-in and M-file methods of that name that are on the MATLAB path and in scope at that time, and stores access information for all of those methods in the handle.

When it comes time to evaluate the function handle, MATLAB considers only those functions that were captured within the handle when it was created. It

is the combination of which functions are in the handle and what arguments the handle is evaluated with that determines which is the actual function that MATLAB dispatches to.

Function handles enable you to do all of the following. Each of these items is explained in more detail in “Benefits of Using Function Handles” in the online documentation:

- Pass function access information to other functions
- Capture all methods of an overloaded function
- Allow wider access to subfunctions and private functions
- Ensure reliability when evaluating functions
- Reduce the number of files that define your functions
- Improve performance in repeated operations
- Manipulate handles in arrays, structures, and cell arrays

You construct a function handle in MATLAB using the *at* sign, @, before the function name. The following example creates a function handle for the `humps` function and assigns it to the variable `fhandle`.

```
fhandle = @humps;
```

You pass the handle in the same way you would pass any argument. This example passes the function handle just created to `fminbnd`, which then minimizes over the interval [0.3, 1].

```
x = fminbnd (fhandle, 0.3, 1)
x =
    0.6370
```

The `fminbnd` function evaluates the `@humps` function handle using the MATLAB `feval` function.

For more information, see “Function Handles” under “Programming and Data Types,” in the MATLAB documentation.

Functions That Operate on Function Handles

The following MATLAB functions now operate on the function handle data type. The `func2str`, `functions`, and `str2func` functions are new to the MATLAB language.

Function	Purpose
`feval`	Evaluate a function through its handle
`func2str`	Construct a function name string from a function handle
`functions`	Display information about a function handle
`isa`	Determine if an object is a function handle
`isequal`	Compare function handles for equality
`str2func`	Construct a function handle from a function name string

CONTINUE Flow Control Statement

The `continue` statement passes control to the next iteration of the `for` or `while` loop in which it appears, skipping any remaining statements in the body of the loop. In nested loops, `continue` passes control to the next iteration of the `for` or `while` loop enclosing it.

The example below shows a `continue` loop that counts the lines of code in the file, `magic.m`, skipping all blank lines and comments. A `continue` statement is used to advance to the next line in `magic.m` without incrementing the count whenever a blank line or comment line is encountered.

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) | strncmp(line,'%',1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines',count));
```

New MATLAB Programming-Related Functions

The following programming-related functions are new in this release.

Function	Purpose
`beep`	Make your computer beep
`genpath`	Generate a path string that includes all directories below a named directory
`iskeyword`	Check if the input string is a MATLAB keyword
`isvarname`	Check if the input string is valid variable name
`nargoutchk`	Validate the number of output arguments
`numel`	Returns the number of elements in an object
`rehash`	Refresh function and file system caches
`support`	Open the MathWorks Technical Support Web page

Creating an Object That Inherits from Parent Classes Only

In MATLAB 5, the `class` function allows you to create a new object that inherits fields and methods from one or more parent classes. However, this new object acquires additional fields that belong to the `structure_name` argument passed in the call to `class`.

The syntax for this command is

```
obj = class(structure_name,'class_name',parent1,parent2...)
```

In MATLAB 6.0, you can create a new object that contains no fields other than those that are inherited from the specified parent objects. Do this using the following syntax.

```
obj = class(struct([]),'class_name',parent1,parent2,...)
```

Code Length Restriction Removed

There is now no limit on the length of a line of M-code.

Running a Syntax Check on M-Files

You can run a check on the syntax of your M-files before executing the files; to do so, use the following command.

```
check_syntactic_warnings
```

This command checks all M-files in the directories specified by the argument list for all warnings that MATLAB generates when reading the M-file into memory. All @class and private directories contained by the argument list directories will also be processed; class and private directories should not be supplied as explicit arguments to this function.

If no argument list is specified, all files on the MATLAB path and in the current working directory will be checked, except those in toolboxes.

If the argument list is exactly `'-toolboxes'`, all files on the MATLAB path and in the current working directory will be checked, including those in toolboxes.

The following command displays the information given above.

```
help check_syntactic_warnings
```

This command can be especially helpful in locating missing separator characters that are now required between array elements. See “Separators Are Now Required Between Array Elements” on page 4-44.

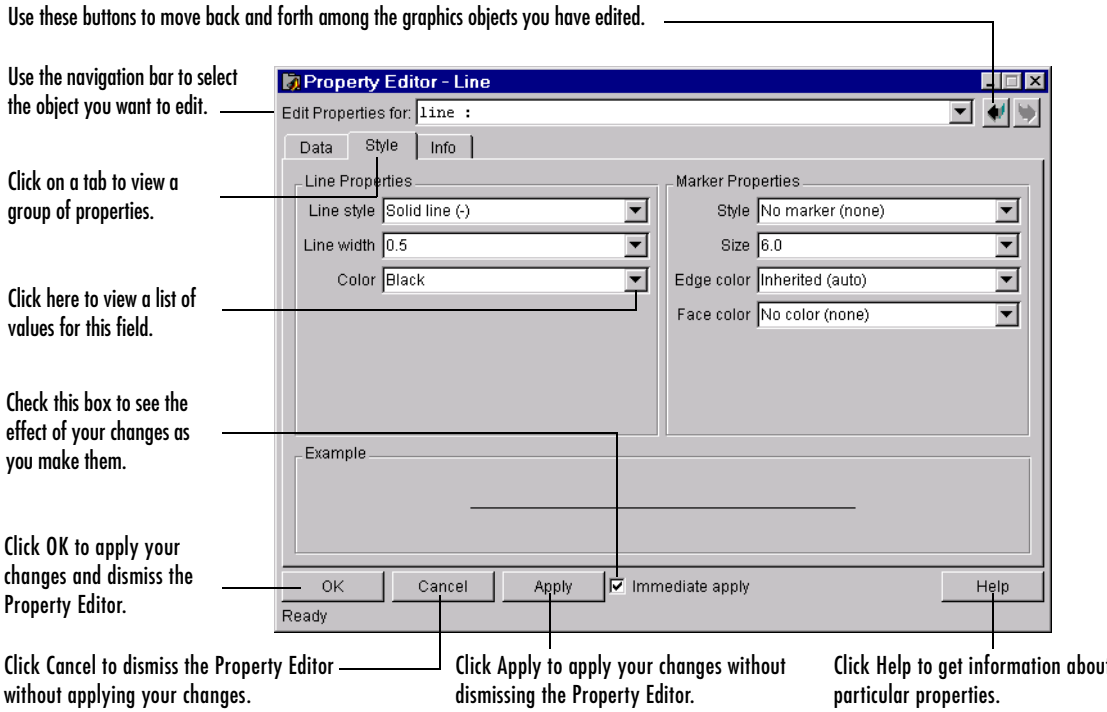
Graphics Features

This section is organized into the following subsections:

Property Editor

MATLAB 6.0 has a new graphical user interface for editing the properties of Handle Graphics objects in MATLAB figures. This tool, called the Property Editor, provides access to many properties of the Handle Graphics objects in a graph, including figures, axes, lines, lights, patches, images, surfaces, rectangles, text, and the root object. Using this tool, you can change the thickness of a line, add titles and axes labels, add lights, and perform many other plot editing tasks.

The following figure shows the components Property Editor interface.



Starting the Property Editor

There are several ways to start the Property Editor.

If plot editing mode is enabled in the figure, you can start the Property Editor by right-clicking on an object and selecting the **Properties** option from the context menu. You can also start the Property Editor by double-clicking on an object in the graph. (Double-clicking on a text object opens a text editing box around the text but does not start the Property Editor.)

If plot editing mode is not enabled, you can start the Property Editor by selecting either the **Figure Properties**, **Axes Properties**, or **Current Object Properties** from the figure window **Edit** menu. These options automatically enable plot editing mode, if it is not already enabled. You can also start the Property Editor from the command line using the `propedit` function.

Note Once you start the Property Editor, you can keep it open throughout an editing session. If you click on another object in the graph, the Property Editor displays the set of panels associated with that object type. You can also use the Property Editor's navigation bar to select an object to edit.

Printing Features

Exporting Figures to the Clipboard. If you use the clipboard to export your figures to graphics-format files, you can now use the **Figure Copy Template Preferences** panel to optimize your figure. You can apply templates specifically for Microsoft Word and PowerPoint, or you can customize your own template.

Page Setup Dialog Box. The **Page Setup** dialog box now presents an enhanced set of options on four tabs.

Tab	Enables you to...
Size and Position	• Print at screen size or set the size of the printed figure • Set the top and left margins and the height and width of the figure • Fill the page, restore the aspect ratio, or center the figure • Choose the units in which the settings are made
Paper	• Choose the paper type or set a custom size • Choose units • Choose paper orientation

Tab	Enables you to... (Continued)
Lines and Text	Set the lines and text in the printed or exported figure to color or black and white
Axis and Figure	• Keep the same settings as are on the screen, or let MATLAB rescale your axes ticks and labels • Print or not print uicontrols that are part of the figure • Keep the screen background color, or let MATLAB change the background color to white • Override the MATLAB default choice of renderer

UNIX Print Dialog Box. The UNIX **Print** dialog box for MATLAB 6.0 has been rewritten. It has many new features available both from the **Print** dialog box and from a new **Options** dialog box:

- From the **Print** dialog box, you can now set the figure size, select a print driver from a scroll list rather than typing it in, and set whether or not you want MATLAB to rescale your axis and ticks limits when it prints your graphic.
- The new **Print Options** dialog box, which launches from the **Print** dialog, enables you to make many settings, including choosing a renderer and setting the printer resolution.

Note Note that you can no longer set the paper orientation or the paper type from the **Print** dialog box; you must use the **Page Setup** dialog box instead.

Preference that Controls Color or Black and White Printing. The **Figure window printing** preference that enables you to set a session-to-session default for sending either black and white or colored lines and text to printers is now located on the **General Preferences** panel.

Note that if your figure does not print correctly in color or black and white according to the figure setting, override the setting from the **File/Preferences** menu. For example, if you specify printing lines and text in color to a color

printer, and you don't get your results in color, go to **File/Preferences** and select **Always send as color**.

PaperSize Property. The figure property `PaperSize` is now writable. You can also set the new `'<custom>'` paper type from the command line.

Note See "Basic Printing and Exporting" in the MATLAB graphics documentation for more information about printing and exporting figures in MATLAB Version 6.0.

3-D Visualization Features

The following table lists new and enhanced volume visualization functions added in MATLAB 6.0.

Function	Purpose
`coneplot`	Create a 3-D coneplot
`contourslice`	Draw contours in slice planes
`curl`	Compute the curl and angular velocity perpendicular to the flow of a 3-D vector field
`divergence`	Compute the divergence of a 3-D vector field
`interpstreamspeed`	Interpolate streamline vertices from speed
`isocolors`	Compute the colors of isosurface vertices
`isosurface`	Extract isosurface
`streamparticles`	Draw stream particles from vector data
`streamribbon`	Draw stream ribbons from vector data
`streamslice`	Draw well-spaced stream lines from vector data

Function	Purpose (Continued)
`streamtube`	Draw stream tubes from vector data
`volumebounds`	Return coordinate and color limits for volume data

Graphics Object Transparency

The transparency of an object determines the degree to which you can see through the object's surface and thereby see other objects that are obscured. You can specify a continuous range of transparency from completely transparent (i.e., invisible) to completely opaque (i.e., no transparency). You can specify transparency properties for the following objects:

- Surface
- Patch
- Images

Transparency Properties. Transparency values, which range from [0 1], are referred to as *alpha* values. MATLAB handles transparency in a way that is analogous to how it handles color:

- Objects can define alpha data that is used as indices into the alphamap or directly as alpha values.
- Objects can define a single face and edge alpha value or use flat or interpolated transparency based on values in the figure's alphamap.
- Axes define alpha limits that control the mapping of object data to alpha values.
- Figures contain alphamaps, which are m-by-1 arrays of alpha values.

The following table summarizes the object properties that control transparency.

Property	Purpose
`ALim`	Alpha axis limits
`ALimMode`	Alpha axis limits mode

Property	Purpose (Continued)
AlphaData	Transparency data for patch, surface, or image
AlphaDataMapping	Transparency data mapping method
Alphamap	Figure alphamap
FaceAlpha	Transparency of the object's faces
EdgeAlpha	Transparency of the object's edges
FaceVertexAlphaData	Patch-only alpha data specification

Transparency Helper Functions. This table lists functions that simplify the process of setting transparency properties.

Function	Purpose
alpha	Set or query transparency properties for objects in current axes
alphamap	Specify the figure alphamap
alim	Set or query the axes alpha limits

See "Transparency" in the MATLAB "3-D Visualization" documentation for more information.

Renderer Autoselection

MATLAB automatically selects the rendering method used based on the content of the figure. If it is available, MATLAB selects OpenGL as the renderer when certain criteria are met. OpenGL is available on many computer systems and is generally faster than the MATLAB other renderers (Painters and Z-buffer). Using OpenGL enables MATLAB to access the graphics hardware that is available on some systems.

Criteria for Autoselection of OpenGL Renderer. When the figure RendererMode property is set to auto, MATLAB uses the OpenGL autoselection criteria to determine whether to select the OpenGL renderer.

When the `RendererMode` property is set to `manual`, MATLAB does not change the renderer, regardless of changes to the figure contents.

If you do not want MATLAB to include OpenGL in the renderer autoselection process, issue the following command.

```
opengl neverselect
```

See the `opengl` reference page for more information.

Camera Toolbar

The Camera Toolbar enables you to perform a number of viewing operations interactively. To display the toolbar, select **Camera Toolbar** from the figure window's **View** menu.

See "View Control with the Camera Toolbar" in the MATLAB documentation for more information.

External Interfaces/API Features

MATLAB Interface to Java

The new Java capability enables you to conveniently bring Java classes into the MATLAB environment, to construct objects from those classes, to call methods on the Java objects, and to save Java objects for later reloading — all accomplished with MATLAB functions and commands. You can also develop your own Java class definitions and make them available for use in MATLAB.

The MATLAB Java interface is intended for all MATLAB users who want to take advantage of the special capabilities of the Java programming language. The interface enables you to

- Access Java application programming interface (API) classes that support essential activities such as I/O and networking. For example, the `URL` class provides convenient access to resources on the internet.
- Easily construct Java objects in MATLAB
- Call Java object methods, using either Java or MATLAB syntax
- Pass data seamlessly between MATLAB variables and Java objects

You construct Java objects in MATLAB by calling the Java class constructor, which has the same name as the class. For example, the following constructor

creates a `Frame` object with the title `'Frame A'`. Any other properties are set to their default values.

```
frame = java.awt.Frame('Frame A');
```

You call methods on this object using either Java syntax

```
frame.setTitle('Sample Frame')
```

Or MATLAB syntax.

```
setTitle(frame,'Sample Frame')
```

You can create n-dimensional arrays of Java objects and call methods on the arrays. You reference and assign to elements of those arrays using MATLAB matrix syntax. You can pass either MATLAB data or Java objects to the methods of Java classes. MATLAB performs data type conversion on this data where necessary.

See “Calling Java from MATLAB” under “External Interfaces/API” in the online help, for full documentation of this feature.

New Functions for the Interface to Java

MATLAB 6.0 provides the following new functions to support the interface to Java.

Function	Purpose
`import`	Add to the current Java packages import list
`isjava`	Test whether an object is a Java object
`javaArray`	Construct a Java array
`javaMethod`	Invoke a Java method (used only in special cases)
`javaObject`	Construct a Java object (used only in special cases)
`methodsview`	Display information on all methods implemented by a Java or MATLAB class

Existing Functions with Java Functionality Added

The following MATLAB functions now operate on Java objects, classes, and methods.

Function	Purpose
`cell`	Convert a Java object array into a MATLAB cell array
`char`	Convert a `java.lang.String` object or array to a `char` array or cell array of `char` arrays
`class`	Return a Java class name
`clear import`	Remove the Java packages import list
`depfun`	Return Java class names that the file and its subordinates use
`disp`	Display a Java object
`double`	Convert a Java object or array to a double array
`exist`	Determine if a Java class exists
`fieldnames`	Return the field names of a Java object
`inmem`	Return the names of loaded Java classes
`isa`	Identify an object as a Java class
`isequal`	Compare Java objects for equality
`methods`	Return all methods of a Java class
`struct`	Convert a Java object or array to a MATLAB structure or structure array
`which`	Return the package, class, and method name for a Java class method

Restriction on Unloading Java Classes

If you load a Java class into MATLAB and, sometime later, modify and recompile the class, you must exit and restart MATLAB in order to use the

updated class definition. This restriction exists because the Java VM does not provide a way to unload a Java class once it has been loaded. Exiting and restarting MATLAB terminates and restarts the Java VM, which then allows you to load the updated class.

Similarly, the `import` function will not import an updated class definition. You must first exit and restart MATLAB.

Java Version

The Java version used by MATLAB differs by platform and for Release 12 is as shown in the following table.

Platform	Java Version
Compaq Alpha	1.1.8-5
IBM	1.2.2
Linux	1.1.8
Microsoft Windows	1.1.8
SGI	1.1.8-00
Sun Solaris	1.1.8-09a

To see the Java version used by MATLAB, type `version -java` in the command window.

New C Language mx Functions

The following `mx` functions are new in this release.

Function	Purpose
`mxCreateNumericMatrix`	Create a numeric matrix and initialize all its data elements to 0
`mxCreateScalarDouble`	Create a scalar, double-precision array initialized to the specified value

Additional Supported Compilers

MATLAB now includes preconfigured options files for these compilers:

- LCC 2.4
- Compaq Fortran 6.1
- Borland's C++Builder 3.0 (Borland C++, Version 5.3)
- Borland's C++Builder 4.0 (Borland C++, Version 5.4)
- Borland's C++Builder 5.0 (Borland C++, Version 5.5)

Note The LCC compiler is included in the MATLAB product set.

Using the Add-In for Visual Studio

The MathWorks provides a MATLAB add-in for the Visual Studio® development system that lets you work easily within Microsoft Visual C/C++ (MSVC). The MATLAB add-in for Visual Studio greatly simplifies using M-files in the MSVC environment. The add-in automates the integration of M-files into Visual C++ projects. It is fully integrated with the MSVC environment.

The add-in for Visual Studio is automatically installed on your system when you run either `mbuild -setup` or `mex -setup` and select Microsoft Visual C/C++ version 5 or 6.

However, there are several steps you must follow in order to use the add-in:

1. To build MEX-files with the add-in for Visual Studio, run the following command at the MATLAB command prompt.

   ```
   mex -setup
   ```

   Follow the menus and choose either Microsoft Visual C/C++ 5.0 or 6.0. This configures `mex` to use the selected Microsoft compiler and also installs the necessary add-in files in your Microsoft Visual C/C++ directories.

2. To configure the MATLAB add-in for Visual Studio to work with Microsoft Visual C/C++:

   a. Select **Tools** -> **Customize** from the MSVC menu.

   b. Click on the **Add-ins and Macro Files** tab.

c Click **Browse**, type `<matlab>\bin\win32` as the filename, and select `Add-ins (.dll)` from the **Files of Type** pulldown list.

Note If Windows is configured to "hide system files," `.dll` files will not be displayed even though they are present on disk. You must change your view options to disable this feature.

---

d Select the `MATLABAddin.dll` file and click **Open**.

e Check **MATLAB for Visual Studio** on the **Add-ins and Macro Files** list and click **Close**. The floating MATLAB add-in for Visual Studio toolbar appears. The checkmark directs MSVC to automatically load the add-in when you start MSVC again.

Note To run the MATLAB add-in for Visual Studio on Windows 98 systems, add this line to your `config.sys` file.

```
shell=c:\command.com /e:32768 /p
```

For additional information on the MATLAB add-in for Visual Studio:

- See the `MATLABAddin.hlp` file in the `<matlab>\bin\win32` directory, or
- Click on the Help icon in the MATLAB add-in for Visual Studio toolbar.

Command Line Override

To specify an option on a one-time basis to the `mex` script, you can use the `mex` command line override feature. For example, the string `/WX` is a Microsoft Visual C/C++ compiler option that causes warnings to be treated as errors. The following statement uses the `mex` command line override feature.

```
mex yprime.c COMPFLAGS#"$COMPFLAGS /WX"
```

The string `COMPFLAGS#"$COMPFLAGS /WX"` tells `mex` to take the default value for `COMPFLAGS` and append `/WX` (space slash W X) to it. To get a list of all

environment variables that can be overridden using this feature, use the command

```
mex -v
```

Any variable name in all uppercase listed in the output can be overridden.

Serial I/O

The MATLAB serial port interface provides direct access to peripheral devices such as modems, printers, and scientific instruments that you connect to your computer's serial port. This interface is established through a serial port object, which you create with the `serial` function. The serial port object supports functions and properties that allow you to

- Configure serial port communications
- Use serial port control pins
- Write and read data
- Use events and actions
- Record information to disk

Note The serial port interface is supported only for Microsoft Windows, Linux, and Sun Solaris platforms.

Creating Graphical User Interfaces – Features

New Graphical User Interface Development Environment

GUIDE, the MATLAB graphical user interface development environment has been redesigned since MATLAB 5.3. The GUIDE toolset for creating graphical user interfaces (GUIs), or visual interfaces, consists of

- Layout Editor – add and arrange objects in the figure window.
- Alignment Tool – align objects with respect to each other.
- Property Inspector – inspect and set property values.
- Object Browser – observe a hierarchical list of the Handle Graphics objects in the current MATLAB session.
- Menu Editor – create window menus and context menus.

You access all of these tools from the Layout Editor. To start the Layout Editor, use the `guide` command.

New Code Architecture

GUIDE now employs a FIG-file to save layout information separately from the M-file that programs the GUI. This approach has several advantages:

- Faster GUI loading
- Generated code provides the framework for application M-files with application options to include a number of useful GUI programming techniques, including:
  - Cross-platform compatibility with respect to GUI figure size, screen location, colors, and fonts
  - Access to object handles without saving global variables or using `findobj`
  - Single or multiple instances of GUI
  - Function prototypes for callback routines
  - GUI switchyard approach to the application program

See “Creating Graphical User Interfaces” in the MATLAB documentation for more information.

Major Bug Fixes

Figure KeyPressFcn

The figure `KeyPressFcn` now generates an event for all keys pressed. Previously, modifiers keys (such as, **Control** and **Esc**) did not generate an event to executed the `KeyPressFcn` callback.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from MATLAB 5.3 (Release 11.0) to MATLAB 6.0. This section is organized into the following subsections:

Development Environment Issues

Preferences

Your preferences from Release 11 are not maintained in Release 12. Specify new preferences for Release 12.

The **Tools** menu in the Release 11 Editor/Debugger, which allowed you to create customized menu items, is no longer supported.

The dos Function Now Returns an Accurate Error Status

The `dos` function now returns a nonzero status when it encounters an error condition. In the past, `dos` always returned zero, indicating success, regardless of whether an error had actually occurred.

Programming and Data Types Issues

Evaluating Function Names

The `feval` function can be used with either function handles or function name strings. Evaluation of function handles is preferred over evaluation of the function name. For example, of the following two lines of code that evaluate the `humps` function, the second is considered to be the preferable mechanism to use.

```
feval('humps', 0.5674);      % Uses a function name string

feval(@humps, 0.5674);       % Uses a function handle
```

To support backward compatibility, `feval` still accepts a function name string as a first argument and evaluates the function named in the string. However,

function handles offer you the additional performance, reliability, and source file control benefits listed in the earlier section “Programming and Data Types Features”.

Argument Name Length for feval, load, and save

Most names in MATLAB are truncated to 31 characters. In previous versions of MATLAB, this was not true for arguments passed to the `feval`, `load`, and `save` functions. As a result, MATLAB was able to distinguish between names passed into these functions when these names were identical up to the thirty first character but unique at some point after that.

In MATLAB 6.0, arguments (other than filenames) that are passed to `feval`, `load`, and `save` are truncated to a maximum of 31 characters. If your code uses names in which the first 31 characters are identical, MATLAB will no longer distinguish between these names within these functions.

Filenames do not hold to this rule. MATLAB uniquely identifies files with names of any length.

Limit on Line Length of M-code Removed

There is no longer any limit on the length of a line of M-code in MATLAB.

feval Accepts Only Simple Function Names

In previous versions of MATLAB, you could call `feval` passing the full path to the function to be evaluated. You could also call a subfunction or private function directly using this method. For example, the following command executes a subfunction that is defined within the file, `subfile`.

```
feval('subfile/mysubfunction')
```

In MATLAB 6.0, `feval` accepts only simple function names. To evaluate subfunctions and private functions from beyond their usual scope, you can use function handles.

Assigning to an Empty Structure Field

Attempting to assign a structure to a field of another structure now results in an error if both of the following conditions are true:

- The field being assigned to has been initialized to an empty matrix.

- The field being assigned to is referenced in the assignment using an array index.

For example:

```
mystruct.emptyfield = [];
mystruct.emptyfield(1) = struct('f1', 25);
??? Conversion to double from struct is not possible.
```

This operation did not return an error in previous versions of MATLAB.

Comparing With an Empty Structure Field

Attempting a string comparison between a string and a structure field now results in an error, if all of the following conditions are true:

- The structure field being compared has been initialized to an empty cell array.
- The structure field being compared is two or more levels down in the structure array.
- The structure array is referenced in the `strcmp` using an array index.

```
mystruct.field1 = struct('field2',{});
strcmp('string', mystruct(1).field1.field2);
??? Error using ==> strcmp
Not enough input arguments.
```

This operation did not return an error in previous versions of MATLAB.

Separators Are Now Required Between Array Elements

MATLAB 5 allowed two or more elements of a cell array, vector, or matrix to appear together without any separating text such as a comma or whitespace, under certain circumstances. For example, the following were all legal expressions.

```
[[1][2]]                    % equivalent to [ 1 2 ]
['hello'sprintf(' world')]  % equivalent to [ 'hello world' ]
{{1}fft(2)}                 % equivalent to { {1} 2 }
```

MATLAB 6.0 issues a warning if it detects this situation upon first loading the M-file into memory. Future versions of MATLAB will likely error at that point

instead. To remedy the situation, place a comma or whitespace between the elements of the cell array, vector or matrix, as in

```
[[1],[2]]
['hello',sprintf(' world')]
{{1},fft(2)}
```

or

```
[[1] [2]]
['hello' sprintf(' world')]
{{1} fft(2)}
```

The following command displays the information given above.

```
help matrix_element_separators
```

Note The section "Running a Syntax Check on M-Files" on page 4-26 describes how MATLAB can help you locate missing element separators in your M-files.

Logical AND and OR Precedence

Starting in MATLAB 6.0, the precedence of the logical AND (&) and logical OR (|) operators now obeys the standard relationship, where AND has a higher precedence than OR. This precedence agrees with the formal rules of Boolean algebra as implemented in most other programming languages, as well as in Simulink and Stateflow.

Previously, MATLAB would incorrectly treat the expression

```
y = a&b | c&d
```

as

```
y = (((a&b) |c) &d);
```

It now correctly treats it as

```
y = (a&b) | (c&d);
```

The only case where the new precedence will impact the result obtained is when | appears before & within the same expression (without parentheses). For example,

```
y = 1 | x & 0;
```

In MATLAB 5.3 and earlier, this statement would yield 0, being evaluated as

```
y = (1 | x) & 0;
```

In MATLAB 6.0 and beyond, this expression yields a 1 as the result, being evaluated as

```
y = 1 | (x & 0);
```

Note We strongly recommend that you add parentheses to all expressions of this form to avoid any potential problems with the interpretation of your code between different versions of MATLAB.

A feature has been provided to allow the system to produce an error for any expression that would change behavior from the old to the new interpretation.

```
feature('OrAndError', 0)  % Warning for any expression that
                          % changed behavior (default)

feature('OrAndError', 1)  % Error for any expression that
                          % changed behavior
```

Breaking From Try-Catch in a Loop

The MATLAB `break` function terminates the execution of a `for` or `while` loop. If a `for` or `while` loop contains a `try` – `catch` statement, with a `break` placed within the `try` – `catch`, the `break` statement terminates the `for` or `while` loop, not the `try` – `catch`.

In the following example, a `break` has been placed in a `try` – `catch` statement and the `try` – `catch` is within a `for` loop. In MATLAB 5, execution of the `break` causes an incorrect branch from the `try` to the line, `x = 5`. The `for` loop is not terminated, in this case.

In MATLAB 6.0, execution of the `break` within the `try` – `catch` terminates the `for` loop. The next line executed following the `break` is `y = 10`.

```
for i = 1:10
        try
            <statement>
            break
        catch
            <statement>
        end
    x = 5               % Break to here in MATLAB 5
    end
y = 10                  % Break to here in MATLAB 6.0
```

Specifying Field Names with setfield and getfield

The `setfield` and `getfield` functions set and get the value of a specified field in a MATLAB structure array. In MATLAB 5, you could use the following undocumented syntaxes to set `fieldn` to the value, `newvalue` and to read back that value.

```
setfield(structurename.field1.fieldn,newvalue)
getfield(structurename.field1.fieldn)
```

These syntaxes are not supported in MATLAB 6.0. You should use the following instead.

```
setfield(structurename,'field1','fieldn',newvalue)
getfield(structurename,'field1','fieldn')
```

For example:

```
a.b.c = 7;

a = setfield(a,'b','c',10);

getfield(a,'b','c')

ans =

    10
```

Using GETFIELD on a Structure Array

The `getfield` function returns the contents of a specified field in a MATLAB structure array. In MATLAB 5, you would get an error if you requested more

than one return value on a single `getfield` call. For example, the following call to `getfield` requests the contents of the `x` field for all three elements of the structure array, `a`.

```
a(1).x=5;
a(2).x=12;
a(3).x=27;

getfield(a,'x')
```

In MATLAB 6.0, requesting more than one output value from `getfield` returns a single output value and does not result in an error. The value returned is the first of all output values requested in the call. In the example above, `getfield` now returns the contents of `a(1).x`, because that is the first field to which `(a,'x')` applies.

```
getfield(a,'x')
ans =
    5
```

You can obtain the contents of another structure array element by specifically indexing to that element. For example:

```
getfield(a,{3},'x')
ans =
    27
```

Temporary Variables Are Anonymous

Individual cells of a cell array, fields of a MATLAB structure, and all temporary variables are now guaranteed to be anonymous.

For example, the following function returns the name of the first input argument passed to `myfun`.

```
function myfun(a)
inputname(1)
```

In MATLAB 5, a temporary variable, used to pass the value of `eval('x')`, retains the variable name, `x`. In MATLAB 6.0, this variable is anonymous.

```
x = [1 2 3];

myfun(eval('x' ))       % Running MATLAB 5, ...
```

```
x                       %   the output is x

myfun(eval('x' ))       % Running MATLAB 6.0, ...
''                      %   the output is ''
```

This change may result in a small difference in the size of some MAT files.

Platform Name Changes for the computer Function

The names returned by the `computer` function for some UNIX platforms have changed for MATLAB 6.0. If you have code that uses the `computer` function for special handling on some UNIX platforms, that code may behave differently because of these platform name changes.

Platform	Pre-6.0 String	String for MATLAB 6.0
Linux	`LNX86`	`GLNX86`
SGI	`SGI`	`SGI` (no change)
SGI64	`SGI64`	`SGI`
HPUX 11.x	`HP700`	`HPUX`
HPUX 10.20	`HP700`	`HP700` (no change)

Processes and Memory Usage on Linux

On Linux, you may notice that MATLAB can start up a large number of processes, even to handle simple tasks. These are actually threads, but due to the “one process per thread” model on Linux, they appear as processes. When you use the ps or top commands in Linux to show the processes running on your system, you will see each of these threads listed as a separate process.

It is not uncommon for Linux to create a large number of threads for an application. Even a simple Java program using AWT and native threads on Linux will create nine threads.

Also displayed is the amount and percentage of memory used by each thread. You should be aware that the memory used by the threads started by MATLAB is shared between threads. The memory consumption reported for each thread in this display actually represents the total memory used by all of the MATLAB threads together.

Obsolete Functions

The following MATLAB functions have either been renamed or have become obsolete. For backwards compatibility, they have not been removed from the language at this time. However, these functions may be removed in a future release, so you are encouraged to discontinue use of the functions, or use the functions that replace them.

Function	Description
`errortrap`	Replace with `try ... catch`
`flops`	Floating-point operation count. Now returns `0`.
`fmin`	Minimize a function of one variable. Replace with `fminbnd`. See instructions for converting your code in "Function Functions" in the MATLAB documentation.
`fmins`	Minimize a function of several variables. Replace with `fminsearch`. See instructions for converting your code in "Function Functions" in the MATLAB documentation.
`foptions`	Default parameters used by the optimization routines. Replace with `optimget`, `optimset`. See instructions for converting your code in "Function Functions" in the MATLAB documentation.
`interp4`, `interp5`, `interp6`	Various two-dimensional data interpolation. Use `interp2` instead.
`isdir`	Replace with `exist`
`isieee`	Return logical true (`1`) on machines with IEEE arithmetic and logical false (`0`) on machines without IEEE arithmetic. Now returns `1` in all cases.
`isstr`	Replace with `ischar`
`meshdom`	Generate `X` and `Y` arrays for three-dimensional plots. Use `meshgrid` instead.

Function	Description (Continued)
`nnls`	Nonnegative least squares. Replace with `lsqnonneg`. See instructions for converting your code in "Function Functions" in the MATLAB documentation.
`quad8`	Numerically evaluate integral. Replaced by `quadl`.
`saxis`	Sound axis scaling. Now has no effect on the output of `sound`.
`setstr`	Replace with `char`
`str2mat`	Replace with `char`
`table1`	One-dimensional table lookup. Use `interp1` instead.
`table2`	Two-dimensional table lookup. Use `interp2` instead.

The errortrap Function Is Disabled But Not Removed

The `errortrap` builtin function is obsolete in MATLAB 6.0. It is disabled, but has not been removed from the language for purposes of backward compatibility.

Specifying Ordinary Differential Equation (ODE) Problems

The ODE problem components that were passed to the solver through an ODE file now are passed directly as arguments or need to be specified in an options structure. The new syntax is

```
solver(@odefun,tspan,y0,options,p1,p2,...)
```

where `odefun`, `tspan` and `y0` are required arguments. See the ODE solver and `odeset` reference pages for details.

MATLAB 6.0 supports use of an ODE file for backwards compatibility, but new functionality is available only with the new syntax.

Output from Background and Foreground Commands (UNIX)

In Release 12 on UNIX platforms, a background command (i.e., any system command after which you add a `&`), such as

```
! cat startup.m &
```

no longer produces any output. Prior to Release 12, a background command sent output to the command window.

If you need to see the output from a command, either do not make the command a background command (i.e., remove the `&`), or run the background command in a separate xterm. To start another xterm, issue the following command.

```
! xterm &
```

In Release 12, foreground functions (i.e., non-background functions) send their output to the diary, if the `diary` function has been issued. The output is also displayed in the command window (prior to Release 12, foreground function output was only displayed in the command window).

matlab_helper Process

To make the `!` and `unix` commands operate more efficiently, in Release 12 MATLAB creates a secondary process, called `matlab_helper`, at startup.

This `matlab_helper` contains those elements of MATLAB necessary to run the `!` and `unix` commands.

External Interfaces/API Issues

Recompile Fortran MEX-Files

Due to changes in the MATLAB/Fortran interface, Fortran MEX-files that were built with versions of MATLAB prior to Release 12 (MATLAB 6.0) will not work unless they are rebuilt with MATLAB 6.0.

MEX File Compatibility

The Release 12 MATLAB Application Program Interface (API) has several compatibility issues relating to existing MEX-files:

- C MEX-files built under MATLAB 4.x may work with MATLAB 6.0, but are no longer supported. You should rebuild these files using MATLAB 6.0.
- You will need to rebuild all MATLAB 5.x C MEX-files on the IBM_RS in MATLAB 6.0 in order to run those MEX-files in MATLAB 6.0.
- MATLAB 6.0 C and Fortran MEX-files will not run on previous versions of MATLAB.

Preference File Location for mex and mbuild

On Windows platforms, the `mex` and `mbuild` commands now look for their preferences file in

```
C:\Winnt\profiles\<user>\application data\mathworks\matlab\R12
```

as opposed to the location used by previous versions.

```
C:\Winnt\profiles\<user>\application data\mathworks\matlab
```

On UNIX platforms, the `mex` and `mbuild` commands now look for their preferences file in

```
/home/<user>/.matlab/R12
```

Therefore, to restore functionality to `mex` and `mbuild`, you will need to either

- Run `mex -setup` and/or `mbuild -setup` (easy way), or
- Manually move your options files from the directory used for preferences in previous releases to the directory now used.

Linux MEX-files

For Release 12, you must rebuild Release 11 MEX-files on Linux.

Due to compatibility issues between versions of the GNU standard C libraries, for Linux the file extension for MEX-files is now `.mexglx`. The file extension for Release 11 was `.mexlx`. This change in file extensions means that the Release 11 MEX-files for Linux will be ignored by Release 12, unless you rebuild the Release 11 MEX-files in Release 12.

SGI MEX-files

You will need to rebuild all MATLAB 5.x C MEX-files on the SGI in MATLAB 6.0 in order to run those MEX-files in MATLAB 6.0. There is no longer a difference between SGI and SGI64 MEX-files; they are all built as `n32`.

Unsupported Compilers

MATLAB no longer supports the following compilers:

- Digital Visual Fortran version 6.0
- Microsoft Visual C/C++ version 4.2

In addition, The MathWorks will drop support for following compilers in a future version of MATLAB. The MathWorks no longer tests with these compilers:

- Watcom C/C++ version 11
- Watcom C/C++ version 10.6

Using the move Command for ActiveX Controls

In previous versions of MATLAB, the `move` command returned its first two output variables in reversed order. In the following command,

```
pos = move(h);
```

the returned value `pos`, was `[y x xsize ysize]` where it should have been `[x y xsize ysize]`.

This has been corrected in MATLAB 6.0. If you have changed your code to accommodate the reversed order in earlier versions of MATLAB, you should correct the order of these variables for MATLAB 6.0.

Return Values for Methods Invoked on ActiveX Objects

In previous releases of MATLAB, invoking a method on an ActiveX object always returned a value of type `double`. Given the example,

```
h = actxcontrol('MWSAMP.MwsampCtrl.1');
val = invoke(h, 'GetI4');
```

instead of returning an `int32` value in `val`, MATLAB converted the type of the object obtained from COM to a `double`.

MATLAB 6.0 returns the same type of object that was passed to it from COM during method invocation. For example, in R12, `val` will be an `int32` value instead of a `double`.

This was done to be consistent with the method signature for a given method. The method signatures for all methods of an interface can be obtained using `invoke` on the handle to an ActiveX object. For example,

```
invoke(h)
```

will list all methods and their signatures for the `mwsamp` control.

If you prefer the former behavior, where MATLAB converts the return value to `double`, you will need to explicitly call `double` on the method parameter you wish to convert. For example,

```
val = double(invoke(h,'GetI4'));
```

yields the old behavior.

mex Function Now Returns Accurate Error Status

The `mex` syntax

```
mex myprog.c
```

now throws an error when it encounters an error condition.

The `mex` syntax

```
stat = mex('myprog.c')
```

now returns a nonzero value to stat when it encounters an error condition.

In the past, on Microsoft Windows platforms, `mex` always either successfully exited or returned zero (indicating success), regardless of whether an error had actually occurred.

To ensure code from before Release 12 works properly in Release 12, either use `try/catch` logic to deal with error conditions, or use a form of `mex` that returns an error status instead of throwing an error. Specifically

```
try
  mex something.c
catch
  disp( something failed );
end
```

or

```
status = mex( something.c );
if status ~= 0
  disp( something failed );
end
```

Using engEvalString with GUI-Intensive Applications

If you have graphical user interface (GUI) intensive applications that execute a lot of callbacks through the MATLAB engine, you should force these callbacks to be evaluated in the context of the base workspace. Use `evalin` to specify that the base workspace is to be used in evaluating the callback expression, as follows.

```
engEvalString(ep,  evalin('base', expression) )
```

Specifying the base workspace in this manner ensures that MATLAB will process the callback correctly and return results for that call.

This does not apply to computational applications that do not execute callbacks.

Starting MATLAB in gdb on GLINUX

If you debug MATLAB using `gdb` on a GLINUX system, your session will stop execution whenever it creates a new thread. You can avoid these interruptions by telling the system not to stop on `SIGCONT` events. You can then proceed to debug your MATLAB code using `nodesktop` mode. You will still be notified whenever a new thread is created, but your session will continue to run without interruption.

Use the following statement in `gdb` to turn off triggering on `SIGCONT` events.

```
handle SIGCONT nostop
```

You can choose to debug in `gdb` without specifying `SIGCONT nostop`, but you will have to type `continue` at each interruption to proceed with your session. The following is a sample debug session in which `SIGCONT nostop` is specified.

```
% matlab -Dgdb

(gdb) handle SIGCONT nostop
Signal Stop Print Pass to program Description
SIGCONT No   Yes   Yes             Continued

(gdb) run -nodesktop
[New Thread 4219]
Program received signal SIGCONT, Continued.
[New Thread 4220]
Program received signal SIGCONT, Continued.
```

MEX-File Extension Changes

The MEX-file extensions have changed for the Linux, SGI, and HP700 platforms for MATLAB 6.0.

Platform	Pre-6.0 Extension	Extension for MATLAB 6.0
Linux	`.mexlx`	`.mexglx`
SGI	`.mexsg`	`.mexsg` (no change)
SGI64	`.mexsg64`	`.mexsg`
HPUX 11.x	`.mexhp7`	`.mexhpux`
HPUX 10.20	`.mexhp7`	`.mexhp7` (no change)

Obsolete C Language MEX Functions

The following API function is obsolete and should not be used in MATLAB programs. This function may not be available in a future version of MATLAB.

```
mexAddFlops
```

Creating Graphical User Interfaces – Upgrade Issues

Editing Version 5 GUIs with Version 6 GUIDE

In Version 5 GUIDE, GUI layouts were saved as MAT-file/M-file pairs. In Version 6, GUIDE saves GUI layouts as FIG-files and M-files. The GUI layout is defined in the FIG-file; there is no generated M-file containing layout information.

Known Software and Documentation Problems

This section updates the MATLAB 6.0 documentation set, reflecting known MATLAB 6.0 software and documentation problems.

This section about software and documentation problems is organized into the following subsections:

Development Environment Problems

Many Open Windows Can Cause a Crash or Hang (Windows 98/Me)

On Microsoft Windows 98/Me platforms, if you keep many windows open, MATLAB may crash or hang. For example, if you keep open about 12 Stateflow windows, or 25 figure windows, or 50 Simulink windows, you may experience this problem. Note that these numbers are only estimates; the actual number of open windows that may cause this problem depends on the resources currently in use by other components and applications.

Maximized Desktop Window Not Remembered on Startup

When you start MATLAB, the desktop configuration is the same as when you last closed MATLAB. However, if the desktop was maximized when you closed MATLAB, it is not maximized upon startup.

Workspace Browser with Over 1000 Variables

If there are over 1000 variables in the workspace and the Workspace browser is open, you might experience performance problems. It is suggested you close the Workspace browser if you expect to have over 1000 variables in the workspace.

Help Browser Doesn't Support Mouse Wheel

The wheel on your mouse will not work in the Help browser.

UNIX Display Problems when UNIX Client and Server Platforms Differ

If you run on UNIX and the platform for the server is different than that for the client, there may be problems with the display of graphics on the client. See the Technical Support Web page for a solution that lists the combinations tested and any known display problems with them.

UNIX Help Browser Search Results Not Highlighted

On UNIX systems, when you perform a full text search using the Help browser, the search returns all of the pages that contain the search term, however the search term is not highlighted when you view a page. To find the term on a page, use the **Find in page** field in the Help browser display pane.

Sun Solaris 16-Bit Display Not Supported

Sun's Java VM for Solaris does not support 16-bit displays. Therefore you cannot use this configuration with Release 12. Use another display mode instead.

Sun Solaris Arrow Keys Not Working

On some Sun Solaris systems, the arrow keys on the main keyboard are not working properly. Instead, try the arrow keys in the numeric keypad.

ALPHA Shortcut Problems When Using Emacs Key Bindings in Editor

On the Alpha platform, if you set the Editor/Debugger preference for key bindings to Emacs, the shortcuts for **Undo** (**Ctrl+_**) and **Copy** (**~+W**) do not work.

Display Problems with Xoftware

If you use Xoftware on a PC to run MATLAB on a UNIX platform, you need to do the following to avoid display problems:

1. Go to the Xoftware **Control Panel**.
2. From the **Options** menu, select **Configuration**.
3. Select the **Window** tab.
4. From the **Options** listing, select `Concurrent Window Manager`.

5 Under **Settings**, select **Off**.

6 Click **OK**.

External Interfaces/API Problems

Available Serial Ports on Windows 98

On Windows 98 platforms, you can access no more than four serial ports. The four serial ports are labeled COM1 through COM4.

The COM1 and COM2 serial ports are standard components of your Windows platform. You can add two additional serial ports, COM3 and COM4. If you have more than four serial ports, you can access only the first four ports. If you try to access a port that does not exist, MATLAB will return the following error message after the `fopen` function is issued.

```
s = serial('COM3');
fopen(s)
??? Error using ==> serial/fopen
Error using ==> fopen
Cannot connect to the COM3 port. Possible reasons are another
application is connected to the port or the port does not exist.
```

Accessing Serial Ports on Solaris

If you repeatedly open and close one or more serial ports on Solaris, MATLAB will become unresponsive. To minimize the chance of encountering this problem, you should:

- Use only one serial port on your platform.
- Connect the serial port object to your device once per MATLAB session.

Graphics Problems

The uimenu Function on Linux

If you use a string containing embedded spaces with the `uimenu` function, MALTAB produces a segmentation violation on the Linux platform. Here is an example that reproduces the problem.

```
hfig = figure;
uimenu = uimenu(hfig, Label ,[ Test  setstr(9)  Menu ]);
```

GUIDE Problems

This section lists know problems with GUIDE:

- If you try to open a file from the **File** menu of the Layout Editor and then cancel the open operation, MATLAB displays an error dialog. Disregard this message.
- On DEC Alpha, there is no Object Browser available. It will be available in the final release of GUIDE.
- On DEC Alpha, figures activated in the Layout Editor are incorrectly sized.

Documentation Updates

Ctrll+Q Quits Without Issuing a Warning

If MATLAB is the active window in your system, using **Ctrl+Q** forces MATLAB to quit without issuing any warning.

interp1 Extrapolation of Out of Range Values

A new argument enables `interp1` to perform extrapolation for out of range values for all methods. It also enables you to specify a scalar to be returned for out of range values.

The PDF version of the `interp1` reference page incorrectly states that the default for all methods is for `interp1` to perform extrapolation for out of range values. In fact, `interp1` performs extrapolation as the default only for the `'spline'`, `'pchip'`, and `'cubic'` methods. For all other methods, it returns `NaN` for out of range values. This behavior is unchanged from Version 5.

The HTML reference page for `interp1` is correct.