Upgrading from MATLAB 4 to MATLAB 5.0

For Use with MATLAB®

Computation

Visualization

Programming

Version 5

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server
`comp.soft-sys.matlab` Newsgroup

`support@mathworks.com` Technical support
`suggest@mathworks.com` Product enhancement suggestions
`bugs@mathworks.com` Bug reports
`doc@mathworks.com` Documentation error reports
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information

Upgrading from MATLAB 4 to MATLAB 5.0

Printing History: January 1998 (online version) New for MATLAB 5.2
October 1998 Revised for MATLAB 5.3 Release 11 (online only)

Contents

1 MATLAB 5.0 Enhancements

2 Upgrading to MATLAB 5.0

Introduction

This manual describes how to upgrade from MATLAB® 4 to MATLAB 5.0.

Who Should Read This Manual?

If you are upgrading from MATLAB 4 to Release 11 (MATLAB 5.3), read this manual first. Then read *Release 11 New Features* for information on how to upgrade from MATLAB 5.0 to Release 11.

If you are upgrading to Release 11 from MATLAB 5.0, 5.1, or 5.2, you do not need to read this manual. You should, instead, read *Release 11 New Features*.

Contents

Chapter 1 describes the new features that were introduced with MATLAB 5.0. New features that were introduced with later releases of MATLAB and its associated products are described in the *Release 11 New Features* document.

Chapter 2 describes the possible code modifications you may need to make to upgrade MATLAB 4 applications to run with MATLAB 5.0. The *Release 11 New Features* document describes how to update from MATLAB 5.0 and later versions to Release 11.

1

MATLAB 5.0 Enhancements

MATLAB 5.0 Enhancements

MATLAB 5.0 featured five major areas of new functionality:

- Enhanced programming and application development tools
- New data types, structures, and language features
- Faster, better graphics and visualization
- More mathematical and data analysis tools
- Major enhancements to the MATLAB application toolbox suite and to Simulink®

Enhanced Programming and Application Development Tools

MATLAB 5.0 provided new M-file programming enhancements and application development tools that make it easier than ever to develop and maintain applications in MATLAB. Highlights include:

- Integrated M-file editor
- Visual M-file debugger
- M-file performance profiler
- Search path browser/editor
- Workspace browser
- Web-based online Help Desk/documentation viewer
- GUI builder
- Handle Graphics® property editor
- Preparsed P-code files (P-files)
- Enhanced, self-diagnosing Application Program Interface (API)

New Data Types, Structures, and Language Features

MATLAB 5.0 introduced new data types and language improvements. These new features make it easy to build much larger and more complex MATLAB applications.

- Multidimensional arrays
- User-definable data structures
- Cell arrays: multitype data arrays
- Character arrays: two bytes per character
- Single byte data type for images
- Object-oriented programming
- Variable-length argument lists
- Multifunction and private M-files
- Function and operator overloading
- `switch`/`case` statements

Faster, Better Graphics and Visualization

MATLAB 5.0 added powerful new visualization techniques and significantly faster graphics using the Z-buffer algorithm. Presentation graphics were also improved to give you more options and control over how you present your data.

- Visualization
  - Truecolor (RGB) support
  - Fast and accurate Z-buffer display algorithm
  - Flat, Gouraud, and Phong lighting
  - Vectorized patches for three dimensional modeling
  - Camera view model, perspective
  - Efficient 8-bit image display
  - Image file import/export

- Presentation graphics
  - Greek symbols, sub/superscripts, multiline text
  - Dual axis plots
  - Three-dimensional quiver, ribbon, and stem plots
  - Pie charts, three-dimensional bar charts
  - Extended curve marker symbol family

More Mathematical and Data Analysis Tools

With more than 500 mathematical, statistical, and engineering functions, MATLAB gives you immediate access to the numeric computing tools you need. New features with MATLAB 5.0 included:

- New ordinary differential equation solvers (ODEs)
- Delaunay triangulation
- Gridding for irregularly sampled data
- Set theory functions
- Two-dimensional quadrature
- Time and date handling functions
- Multidimensional interpolation, convolution, and FFT's
- Bit-wise operators
- Iterative sparse methods
- Sparse matrix eigenvalues and singular values

Enhancements to Application Toolboxes and to Simulink

Significant upgrades to application toolboxes and Simulink introduced with MATLAB 5.0 were:

- Simulink 2.0
- Image Processing Toolbox 2.0
- Control System Toolbox 4.0
- Signal Processing Toolbox 4.0
- Optimization Toolbox 2.0

New Data Constructs

MATLAB 5.0 added support for these new data constructs:

- Multidimensional arrays
- Cell arrays
- Structures
- Objects

In addition, MATLAB 5.0 featured character arrays that incorporate an improved storage method for string data.

Multidimensional Arrays

Arrays (other than sparse matrices) are no longer restricted to two dimensions. You can create and access arrays with two or more dimensions by:

- Using MATLAB functions like `zeros`, `ones`, or `rand`
- Using the new `cat` function
- Using the `repmat` function
- Indexing an existing array

MATLAB functions like `zeros`, `ones`, and `rand` were extended to accept more than two dimensions as arguments. To create a 3-by-4-by-5 array of ones, for example, use

```
A = ones(3,4,5)
```

The new `cat` function enables you to concatenate arrays along a specified dimension. For example, create two rectangular arrays `A` and `B`:

```
A = [1 2 3; 4 5 6];
B = [6 2 0; 9 1 3];
```

To concatenate these along the third dimension:

```
C = cat(3,A,B)

C(:,:,1) =

     1     2     3
     4     5     6


C(:,:,2) =

     6     2     0
     9     1     3
```

You can also create an array with two or more dimensions in which every element has the same value using the `repmat` function. `repmat` accepts the value with which to fill the array, followed by a vector of dimensions for the array. For example, to create a 2-by-2-by-3-by-3 array `B` where every element has the value `pi`:

```
B = repmat(pi,[2 2 3 3]);
```

You can also use `repmat` to replicate or "tile" arrays in a specified configuration.

Table 1-1: New Multidimensional Array Functions

Function	Description
`cat`	Concatenate arrays.
`flipdim`	Flip array along specified dimension.
`ndgrid`	Generate arrays for multidimensional functions and interpolation.
`ndims`	Return number of array dimensions.
`permute, ipermute`	Permute the dimensions of a multidimensional array.
`reshape`	Change size.
`shiftdim`	Shift dimensions.

Table 1-1: New Multidimensional Array Functions (Continued)

Function	Description
`squeeze`	Remove singleton array dimensions.
`sub2ind, ind2sub`	Return single index from subscripts; subscripts from linear index.

Cell Arrays

Cell arrays have elements that are containers for any type of MATLAB data, including other cells. You can build cell arrays using:

- The cell array constructor `{}`
- Assignment statements (for instance, `A{2,2} = 'string'`)
- The new `cell` function

Table 1-2: New Cell Array Functions

Function	Description
`cell`	Create a cell array.
`cell2struct`	Convert cell array to structure array.
`celldisp`	Display top-level structure of a cell array.
`cellplot`	Graphically display the structure of a cell array.
`num2cell`	Convert a matrix into a cell array.

Structures

Structures are constructs that have named fields containing any kind of data. For example, one field might contain a text string representing a name (`patient.name = 'Jane Doe'`), another might contain a scalar representing a billing amount (`patient.billing = 127.00`), and a third might hold a matrix of medical test results. You can organize these structures into arrays of data.

Create structure arrays by using individual assignment statements or the new `struct` function.

Table 1-3: New Structure Functions

Function	Description
`fieldnames`	Return field names of structure array.
`getfield`	Get field of structure array.
`rmfield`	Remove structure fields.
`setfield`	Set field of structure array.
`struct`	Create structure array.
`struct2cell`	Convert structure to a cell array.

MATLAB Objects

Object-oriented programming is now available within the MATLAB environment.

Objects

The MATLAB programming language does not require the use of data types. For many applications, however, it is helpful to associate specific attributes with certain categories of data. To facilitate this, MATLAB allows you to work with *objects*. Objects are typed structures. A single *class* name identifies both the type of the structure and the name of the function that creates objects belonging to that class.

Objects differ from ordinary structures in two important ways:

Data hiding. The structure fields of objects are not visible from the command line. Instead, you can access structure fields only from within a *method*, an M-file associated with the object class. Methods reside in class directories. Class directories have the same name as the class, but with a prepended `@` symbol. For example, a class directory named `@inline` might contain methods for a class called `inline`.

Function and expression overloading. You can create methods that override existing M-files. If an object calls a function, MATLAB first checks to see if there is a method of that name before calling a supplied M-file of that name. You can also provide methods that are called for MATLAB operators. For objects `a` and `b`, for instance, the expression `a + b` calls the method `plus(a,b)` if it exists

Table 1-4: New Object-Oriented Functions

Function	Description
`class`	Create or return class of object.
`isa`	True if object is a given class.
`inferiorto`	Inferior class relationship.
`superiorto`	Superior class relationship.

Character Arrays

Strings now take up less memory than they did in previous releases. MATLAB 4 required 64 bits per character for string data. MATLAB 5.0 reduced the memory required to only 16 bits per character.

Table 1-5: New Character String Functions

Function	Description
`base2dec`	Convert base `B` to decimal number.
`bin2dec`	Convert binary to decimal number.
`char`	Convert numeric values to string.
`dec2base`	Convert decimal number to base.
`dec2bin`	Convert decimal to binary number.
`mat2str`	Convert a matrix into a string.
`strcat`	String concatenation.
`strmatch`	Find possible matches for a string.

Table 1-5: New Character String Functions (Continued)

Function	Description
`strncmp`	Compare the first n characters of two strings.
`strvcat`	Vertical concatenation of strings.

Programming Capabilities

MATLAB 5.0 included flow-control improvements and new M-file programming tools.

Flow-Control Improvements

MATLAB 5.0 featured:

- A new flow-control statement, the `switch` statement
- More efficient evaluation of `if` expressions

The `switch` statement is a convenient way to execute code conditionally when you have many possible cases to choose from. It is no longer necessary to use a series of `elseif` statements:

```
switch input_num
    case –1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
         disp('other value');
end
```

Only the first matching `case` is executed.

`switch` can handle multiple conditions in a single `case` statement by enclosing the `case` expression in a cell array. For example, assume `method` exists as a string variable:

```
switch lower(method)
    case {'linear','bilinear'}, disp('Method is linear')
    case 'cubic', disp('Method is cubic')
    case 'nearest', disp('Method is nearest')
    otherwise, disp('Unknown method.')
end
```

Table 1-6: New Flow Control Commands

Command	Description
`case`	Case switch.
`otherwise`	Default part of `switch` statement.
`switch`	Conditionally execute code, switching among several cases.

MATLAB now evaluates `if` expressions more efficiently than before. For example, consider the expression `if a|b`. If `a` is true, then MATLAB will not evaluate `b`. Similarly, MATLAB won't execute statements following the expression `if a&b` in the event `a` is found to be false.

Table 1-7: New Logical Operators

Operator	Description
`iscell`	True for a cell array.
`isequal`	True if arrays are equal.
`isfinite`	True for finite elements.
`islogical`	True for logical arrays.
`isnumeric`	True if input is a numeric array.
`isstruct`	True for a structure.
`logical`	Convert numeric values to logical vectors.

M-File Programming Tools

MATLAB 5.0 added four features to enhance MATLAB's M-file programming capabilities.

Variable Number of Input and Output Arguments

The `varargin` and `varargout` commands simplify the task of passing data into and out of M-file functions. For instance, the statement `function varargout = myfun(A,B)` allows M-file `myfun` to return an arbitrary number of output arguments, while the statement `function [C,D] = myfun(varargin)` allows it to accept an arbitrary number of input arguments.

Multiple Functions Within an M-File

It is now possible to have subfunctions within the body of an M-file. These are functions that the primary function in the file can access but that are otherwise invisible.

M-File Profiler

This utility lets you debug and optimize M-files by tracking cumulative execution time for each line of code. Whenever the specified M-file executes, the profiler counts how many time intervals each line uses.

Pseudocode M-Files

The `pcode` command saves a pseudocode version of a function or script to disk for later sessions. This *pseudocode* version is ready-to-use code that MATLAB can access whenever you invoke the function.

Table 1-8: New Programming Tools

Function	Description
`addpath`	Append directory to MATLAB's search path.
`assignin`	Assign variable in workspace.
`edit`	Edit an M-file.
`editpath`	Modify current search path.
`evalin`	Evaluate variable in workspace.

Table 1-8: New Programming Tools (Continued)

Function	Description
`fullfile`	Build full filename from parts.
`inmem`	Return functions in memory.
`inputname`	Return input argument name.
`mfilename`	Return name of the currently running M-file.
`mexext`	Return the MEX filename extension.
`pcode`	Create pseudocode file (P-file).
`profile`	Measure and display M-file execution profiles.
`rmpath`	Remove directories from MATLAB's search path.
`varargin`, `varargout`	Pass or return variable numbers of arguments.
`warning`	Display warning message.
`web`	Point Web browser at file or Web site.

New and Enhanced Language Functions

MATLAB 5.0 provided a large number of new language functions as well as enhancements to existing functions.

Table 1-9: New Elementary and Specialized Math Functions

Function	Description
`airy`	Airy functions.
`besselh`	Bessel functions of the third kind (Hankel).
`condeig`	Condition number with respect to eigenvalues.
`condest`	1-norm matrix condition estimate.
`dblquad`	Numerical double integration
`mod`	Modulus (signed remainder after division).
`normest`	2-norm estimate.

Table 1-10: New Time and Date Functions

Function	Description
`calendar`	Calendar.
`datenum`	Serial date number.
`datestr`	Create date string.
`datetick`	Date formatted tick labels.
`datevec`	Date components.
`eomday`	End of month.
`now`	Current date and time.
`weekday`	Day of the week.

Table 1-11: New Ordinary Differential Equation Functions

Function	Description
`ode45`, `ode23`, `ode113`, `ode23s`, `ode15s`	Solve differential equations, low- and high- order methods.
`odefile`	Define a differential equation problem for ODE solvers.
`odeget`	Extract options from an argument created with `odeset`.
`odeset`	Create and edit input arguments for ODE solvers.

Table 1-12: New Matrix Functions

Function	Description
`cholinc`	Incomplete Cholesky factorization.
`gallery`	More than 50 new test matrices.
`luinc`	Incomplete LU factorization.
`repmat`	Replicate and tile an array.
`sprand`	Random uniformly distributed sparse matrices.

Table 1-13: New Methods for Sparse Matrices

Method	Description
`bicg`	BiConjugate Gradients method.
`bicgstab`	BiConjugate Gradients Stabilized method.
`cgs`	Conjugate Gradients Squared method.
`eigs`	Find a few eigenvalues and eigenvectors.
`gmres`	Generalized Minimum Residual method.

Table 1-13: New Methods for Sparse Matrices (Continued)

Method	Description
pcg	Preconditioned Conjugate Gradients method.
qmr	Quasi-Minimal Residual method.
svds	A few singular values.

Subscripting and Assignment Enhancements

In MATLAB 5.0, you can:

- Access the last element of an array using the end keyword.
- Obtain consistent results for indexing expressions consisting of all ones.
- Use scalar expansion in subarray assignments.

A statement like A(ones([m,n])) now always returns an m-by-n array in which each element is A(1). In previous versions, the statement returned different results depending on whether A was or was not an m-by-n matrix.

In previous releases, expressions like A(2:3,4:5) = 5 resulted in an error. MATLAB 5.0 automatically "expands" the 5 to be the right size (that is, 5*ones(2,2)).

Integer Bit Manipulation Functions

The ops directory contains commands that permit bit-level operations on integers. Operations include setting and unsetting, complementing, shifting, and logical AND, OR, and XOR.

Table 1-14: New Bitwise Functions

Function	Description
bitand	Bitwise AND.
bitcmp	Complement bits.
bitget	Get bit.
bitmax	Maximum floating-point integer.

Table 1-14: New Bitwise Functions (Continued)

Function	Description
bitor	Bitwise OR.
bitset	Set bit.
bitshift	Bitwise shift.
bitxor	Bitwise XOR.

Dimension Specification for Data Analysis Functions

MATLAB's basic data analysis functions now enable you to supply a second input argument. This argument specifies the dimension along which the function operates. For example, create an array A:

```
A = [3 2 4; 1 0 5; 8 2 6];
```

To sum along the first dimension of A, incrementing the row index, specify 1 for the dimension of operation:

```
sum(A,1)

ans =

    12     4    15
```

To sum along the second dimension, incrementing the column index, specify 2 for the dimension:

```
sum(A,2)

ans =

     9
     6
    16
```

Other functions that accept the dimension specifier include prod, cumprod, and cumsum.

Wildcards in Utility Commands

The asterisk (*) can be used as a wildcard in the `clear` and `whos` commands. This allows you, for example, to clear only variables beginning with a given character or characters, as in

```
clear A*
```

Empty Arrays

Earlier versions of MATLAB allowed for only one empty matrix, the 0-by-0 matrix denoted by `[]`. MATLAB 5.0 provided for matrices and arrays in which some, but not all, of the dimensions are zero. For example, `1-by-0`, `10-by-0-by-20`, and `[3 4 0 5 2]` are all possible array sizes.

The two-character sequence `[]` continues to denote the 0-by-0 matrix. Empty arrays of other sizes can be created with the functions `zeros`, `ones`, `rand`, or `eye`. To create a 0-by-5 matrix, for example, use

```
E = zeros(0,5)
```

The basic model for empty matrices is that any operation that is defined for `m`-by-`n` matrices, and that produces a result with a dimension that is some function of `m` and `n`, should still be allowed when `m` or `n` is zero. The size of the result should be that same function, evaluated at zero.

For example, horizontal concatenation

```
C = [A B]
```

requires that `A` and `B` have the same number of rows. So if `A` is `m`-by-`n` and `B` is `m`-by-`p`, then `C` is `m`-by-`(n+p)`. This is still true if `m` or `n` or `p` is zero.

Many operations in MATLAB produce row vectors or column vectors. It is now possible for the result to be the empty row vector

```
r = zeros(1,0)
```

or the empty column vector

```
c = zeros(0,1)
```

Some MATLAB functions, like `sum` and `max`, are *reductions*. For matrix arguments, these functions produce vector results; for vector arguments they produce scalar results. Backwards compatibility issues arise for the argument `[]`, which in MATLAB 4 played the role of both the empty matrix and the empty vector. In MATLAB 5.0, empty inputs with these functions produce these results:

- `sum([])` is `0`
- `prod([])` is `1`
- `max([])` is `[]`
- `min([])` is `[]`

New Data Analysis Features

MATLAB 5.0 provided an expanded set of basic data analysis functions.

Table 1-15: New Statistical Data Analysis Functions

Function	Description
`convhull`	Convex hull.
`cumtrapz`	Cumulative trapezoidal numerical integration.
`delaunay`	Delaunay triangularization.
`dsearch`	Search for nearest point.
`factor`	Prime factors.
`inpolygon`	Detect points inside a polygonal region.
`isprime`	True for prime numbers.
`nchoosek`	All possible combinations of `n` elements taken `k` at a time.
`perms`	All possible permutations.
`polyarea`	Area of polygon.
`primes`	Generate a list of prime numbers.
`sortrows`	Sort rows in ascending order.
`tsearch`	Search for enclosing Delaunay triangle.
`voronoi`	Voronoi diagram.

MATLAB 5.0 also offered expanded data analysis in the areas of:

- Higher-dimension interpolation
- Extended `griddata` functionality based on Delaunay triangulation
- New set theoretic functions

Higher-Dimension Interpolation

The new functions `interp3` and `interpn` let you perform three-dimensional and multidimensional interpolation. `ndgrid` provides arrays that can be used in multidimensional interpolation.

Table 1-16: New Interpolation Functions

Function	Description
`interp3`	Three-dimensional data interpolation (table lookup).
`interpn`	Multidimensional data interpolation (table lookup).
`ndgrid`	Generate arrays for multidimensional functions and interpolation.

griddata Based on Delaunay Triangulation

`griddata` supports triangle-based interpolation using nearest neighbor, linear, and cubic techniques. It creates smoother contours on scattered data using the cubic interpolation method.

Set Theoretic Functions

The functions `union`, `intersect`, `ismember`, `setdiff`, and `unique` treat vectors as sets, allowing you to perform operations like union $A \cup B$, intersection $A \cap B$, and difference $A - B$ of such sets. Other set-theoretical operations include location of common set elements (`ismember`) and elimination of duplicate elements (`unique`).

Table 1-17: New Set Functions

Function	Description
`intersect`	Set intersection of two vectors.
`ismember`	Detect members of a set.
`setdiff`	Return the set difference of two vectors.
`setxor`	Set `XOR` of two vectors.

Table 1-17: New Set Functions (Continued)

Function	Description
`union`	Set union of two vectors.
`unique`	Return unique elements of a vector.

New and Enhanced Handle Graphics Features

MATLAB 5.0 provided significant improvements to Handle Graphics. For details on MATLAB graphics features, see *Using MATLAB Graphics.*

Plotting Capabilities

MATLAB's basic plotting capabilities have been improved and expanded in MATLAB 5.0.

Table 1-18: New and Enhanced Plotting Capabilities

Function	Description
`area`	Filled area plot.
`bar3`	Vertical 3-D bar chart.
`bar3h`	Horizontal 3-D bar chart.
`barh`	Horizontal bar chart.
`pie`	Pie chart.
`pie3`	Three-dimensional pie chart.
`plotyy`	Plot graphs with Y tick labels on left and right.

Filling Areas

The `area` function plots a set of curves and fills the area beneath the curves.

Bar Chart Enhancements

`bar3`, `bar3h`, and `barh` draw vertical and horizontal bar charts. These functions, together with `bar`, support multiple filled bars in grouped and stacked formats.

Labels for Patches and Surfaces

`legend` can label any solid-color patch and surface. You can now place legends on line, bar, ribbon, and pie plots, for example.

Table 1-19: New Graph Annotation Functions

Function	Description
`box`	Axes box.
`datetick`	Display dates for axes tick labels.

Marker Style Enhancement

A number of new line markers are available, including, among others, a square, a diamond, and a five-pointed star. These can be specified independently from line style.

Stem Plot Enhancements

`stem` and `stem3` plot discrete sequence data as filled or unfilled stem plots.

Three-Dimensional Plotting Support

`quiver3` displays three-dimensional velocity vectors with (`u`,`v`,`w`) components. The `ribbon` function displays data as three-dimensional strips.

Table 1-20: New Three-Dimensional Plotting Functions

Function	Description
`quiver3`	Three-dimensional quiver plot.
`ribbon`	Draw lines as 3-D strips.
`stem3`	Three-dimensional stem plot.

Data Visualization

MATLAB 5.0 introduced many new and enhanced capabilities for data visualization.

New Viewing Model

Axes camera properties control the orthographic and perspective view of the scene created by an axes and its child objects. You can view the axes from any location around or in the scene, as well as adjust the rotation, view angle, and target point.

New Method for Defining Patches

You can define a patch using a matrix of faces and a matrix of vertices. Each row of the face matrix contains indices into the vertex matrix that define the connectivity of the face. Defining patches in this way reduces memory consumption because you no longer need to specify redundant vertices.

Triangular Meshes and Surfaces

The new functions `trimesh` and `trisurf` create triangular meshes and surfaces from `x`, `y`, and `z` vector data and a list of indices into the vector data.

Table 1-21: New Triangular Mesh and Surface Functions

Function	Description
`trisurf`	Triangular surface plot.
`trimesh`	Triangular mesh plot.

Improved Slicing

`slice` now supports an arbitrary slicing surface.

Contouring Enhancements

The contouring algorithm now supports parametric surfaces and contouring on triangular meshes. In addition, `clabel` rotates and inserts labels in contour plots.

Table 1-22: New Contour Plot

Function	Description
`contourf`	Filled contour plot.

New zoom Options

The `zoom` function supports two new options:

- `scale_factor` – zooms by the specified scale factor relative to the current zoom state (e.g., `zoom(2)` zooms in by a factor of two).
- `fill` – zooms to the point where the objects contained in the axes are as large as they can be without extending beyond the axes plot box from any view. Use this option when you want to rotate the axes without seeing an apparent size change.

Graphics Presentation

MATLAB 5.0 provided improved control over the display of graphics objects.

Enhancements to Axes Objects

MATLAB 5.0 added more advanced control for three-dimensional axes objects. You can control the three-dimensional aspect ratio for the axes' plot box, as well as for the data displayed in the plot box. You can also zoom in and out from a three-dimensional axes using viewport scaling and axes camera properties.

The `axis` command supports a new option designed for viewing graphics objects in 3-D:

```
axis vis3d
```

This option prevents MATLAB from stretching the axes to fit the size of the Figure window and otherwise altering the proportions of the objects as you change the view.

In a two-dimensional view, you can display the *x*-axis at the top of an axes and the *y*-axis at the right side of an axes.

Color Enhancements

`colordef white` or `colordef black` changes the color defaults on the root so that subsequent figures produce plots with a white or black axes background color. The figure background color is changed to be a shade of gray, and many other defaults are changed so that there will be adequate contrast for most plots. `colordef none` sets the defaults to their MATLAB 4 values. In addition, a number of new colormaps are available.

Table 1-23: New Figure and Axis Color Control

Function	Description
`colordef`	Select figure color scheme.

Table 1-24: New Colormaps

Function	Description
`autumn`	Shades of red and yellow colormap.
`colorcube`	Regularly spaced colors in RGB colorspace, plus more steps of gray, pure red, pure green, and pure blue.
`lines`	Colormap of colors specified by the axes' `ColorOrder` property.
`spring`	Shades of magenta and yellow colormap.
`summer`	Shades of green and yellow colormap.
`winter`	Shades of blue and green colormap.

Text Object Enhancements

MATLAB 5.0 supports a subset of TeX commands. A single text graphics object can support multiple fonts, subscripts, superscripts, and Greek symbols. See the `text` function in the online MATLAB Function Reference for information about the supported TeX subset.

You can also specify multiline character strings and use normalized font units so that text size is a fraction of an axes' or uicontrol's height. MATLAB supports multiline text strings using cell arrays. Simply define a string variable as a cell array with one line per cell.

Improved General Graphics Features

The MATLAB startup file sets default properties for various graphics objects so that new figures are aesthetically pleasing and graphs are easier to understand.

Table 1-25: New Figure Window Creation and Control Command

Command	Description
`dialog`	Create a dialog box.

Z-buffering is now available for fast and accurate three-dimensional rendering.

MATLAB 5.0 provided built-in menus on X Window systems. `Figure MenuBar 'figure'` is now supported on UNIX.

Lighting

MATLAB added support for a new graphics object called a light. You create a light object using the `light` function. Three important light object properties are:

- `Color` – the color of the light cast by the light object
- `Style` – either infinitely far away (the default) or local
- `Position` – the direction (for infinite light sources) or the location (for local light sources)

You cannot see light objects themselves, but you can see their effect on any patch and surface objects present in the same axes. You can control these effects by setting various patch and surface object properties. `AmbientStrength`, `DiffuseStrength`, and `SpecularStrength` control the intensity of the respective light-reflection characteristics; `SpecularColorReflectance` and `SpecularExponent` provide additional control over the reflection characteristics of specular light.

The Axes `AmbientLightColor` property determines the color of the ambient light, which has no direction and affects all objects uniformly. Ambient light effects occur only when there is a visible light object in the axes.

The light object's `Color` property determines the color of the directional light, and its `Style` property determines whether the light source is a point source (`Style` set to `local`), which radiates from the specified position in all directions, or a light source placed at infinity (`Style` set to `infinite`), which shines from the direction of the specified position with parallel rays.

You can also select the algorithm used to calculate the coloring of the lit objects. The patch and surface `EdgeLighting` and `FaceLighting` properties select between no lighting, and flat, Gouraud, or Phong lighting algorithms.

print Command Revisions

The `print` command was extensively revised for MATLAB 5.0. Consult *Using MATLAB Graphics* for a complete description of `print` command capabilities. Among the new options available for MATLAB 5.0:

- The `–loose` option makes the PostScript bounding box equal to the figure's `PaperPosition` property. EPSI (X Window systems) previews are the same size as the generated PostScript drawing.
- Z-buffer images may be printed at user-selectable resolution.
- The `–dmeta` option now supports Enhanced Windows Metafiles.
- `print –dmfile` generates an M-file that recreates a figure.
- Uicontrol objects print by default unless suppressed with the `–noui` option. In earlier versions of MATLAB, uicontrols did not appear when you printed figures. If you specify the `–noui` option with the `print` command, MATLAB ignores uicontrols and prints only axes and axes children.

Additional print Device Options

The `print` command has several new device options.

Table 1-26: print Command Device Options

Device	Description
`–dljet4`	HP LaserJet 4 (defaults to 600 dpi)
`–ddeskjet`	HP DeskJet and DeskJet Plus

Table 1-26: print Command Device Options (Continued)

Device	Description
-ddjet500	HP Deskjet 500
-dcdj500	HP DeskJet 500C
-dcdj550	HP Deskjet 550C
-dpjxl	HP PaintJet XL color printer
-dpjxl300	HP PaintJet XL300 color printer
-ddnj650c	HP DesignJet 650C
-dbj200	Canon BubbleJet BJ200
-dbjc600	Canon Color BubbleJet BJC-600 and BJC-4000
-dibmpro	IBM 9-pin Proprinter
-dbmp256	8-bit (256-color) BMP file format
-dbmp16m	24-bit BMP file format
-dpcxmono	Monochrome PCX file format
-dpcx24b	24-bit color PCX file format, three 8-bit planes
-dpbm	Portable Bitmap (plain format)
-dpbmraw	Portable Bitmap (raw format)
-dpgm	Portable Graymap (plain format)
-dpgmraw	Portable Graymap (raw format)
-dppm	Portable Pixmap (plain format)
-dppmraw	Portable Pixmap (raw format)

Image Support

MATLAB 5.0 made a number of enhancements to image support. These enhancements include:

- Truecolor support
- New functions for reading images from and writing images to graphics files
- 8-bit image support

Truecolor

In addition to indexed images, in which colors are stored as an array of indices into a colormap, MATLAB 5.0 now supports truecolor images. A truecolor image does not use a colormap; instead, the color values for each pixel are stored directly as RGB triplets. In MATLAB, the `CData` property of a truecolor image object is a three-dimensional (`m`-by-`n`-by-3) array. This array consists of three `m`-by-`n` matrices (representing the red, green, and blue color planes) concatenated along the third dimension.

Reading and Writing Images

The `imread` function reads image data into MATLAB arrays from graphics files in various standard formats, such as TIFF. You can then display these arrays using the `image` function, which creates a Handle Graphics image object. You can also write MATLAB image data to graphics files using the `imwrite` function. `imread` and `imwrite` both support a variety of graphics file formats and compression schemes.

8-Bit Images

When you read an image into MATLAB using `imread`, the data is stored as an array of 8-bit integers. This is a much more efficient storage method than the double-precision (64-bit) floating-point numbers that MATLAB typically uses.

The Handle Graphics image object has been enhanced to support 8-bit `CData`. This means you can display 8-bit images without having to convert the data to double precision. MATLAB 5.0 also supports a limited set of operations on these 8-bit arrays. You can view the data, reference values, and reshape the array in various ways. To perform any mathematical computations, however, you must first convert the data to double precision, using the `double` function.

Note that, in order to support 8-bit images, certain changes have been made in the way MATLAB interprets image data. This table summarizes the conventions MATLAB uses:

Image Type	Double-Precision Data (Double Array)	8-Bit Data (uint8 Array)
Indexed (colormap)	Image is stored as a 2-D (`m`-by-`n`) array of integers in the range [`1`,`length(colormap)`]; colormap is an `m`-by-3 array of floating-point values in the range [0, 1].	Image is stored as a 2-D (`m`-by-`n`) array of integers in the range [0, 255]; colormap is an `m`-by-3 array of floating-point values in the range [0, 1]
Truecolor (RGB)	Image is stored as a 3-D (`m`-by-`n`-by-3) array of floating-point values in the range [0, 1].	Image is stored as a 3-D (`m`-by-`n`-by-3) array of integers in the range [0, 255].

Note that MATLAB interprets image data very differently depending on whether it is double precision or 8-bit. The rest of this section discusses things you should keep in mind when working with image data to avoid potential pitfalls. This information is especially important if you want to convert image data from one format to another.

Indexed Images

In an indexed image of class `double`, the value 1 points to the first row in the colormap, the value 2 points to the second row, and so on. In a `uint8` indexed image, there is an offset; the value 0 points to the first row in the colormap, the value 1 points to the second row, and so on. The `uint8` convention is also used in graphics file formats, and enables 8-bit indexed images to support up to 256 colors. Note that when you read in an indexed image with `imread`, the resulting image array is always of class `uint8`. (The colormap, however, is of class `double`; see below.)

If you want to convert a `uint8` indexed image to `double`, you need to add 1 to the result. For example:

```
X64 = double(X8) + 1;
```

To convert from double to uint8, you need to first subtract 1, and then use round to ensure all the values are integers:

```
X8 = uint8(round(X64 – 1));
```

The order of the operations must be as shown in these examples, because you cannot perform mathematical operations on uint8 arrays.

When you write an indexed image using imwrite, MATLAB automatically converts the values if necessary.

Colormaps

Colormaps in MATLAB are always m-by-3 arrays of double-precision floating-point numbers in the range [0, 1]. In most graphics file formats, colormaps are stored as integers, but MATLAB does not support colormaps with integer values. imread and imwrite automatically convert colormap values when reading and writing files.

Truecolor Images

In a truecolor image of class double, the data values are floating-point numbers in the range [0, 1]. In a truecolor image of class uint8, the data values are integers in the range [0, 255].

If you want to convert a truecolor image from one data type to the other, you must rescale the data. For example, this call converts a uint8 truecolor image to double:

```
RGB64 = double(RGB8)/255;
```

This call converts a double truecolor image to uint8:

```
RGB8 = uint8(round(RGB*255));
```

The order of the operations must be as shown in these examples, because you cannot perform mathematical operations on uint8 arrays. When you write a truecolor image using imwrite, MATLAB automatically converts the values if necessary.

New and Enhanced Handle Graphics Object Properties

This section lists new graphics object properties supported in MATLAB 5.0. It also lists graphics properties whose behavior has changed significantly. *Using MATLAB Graphics* and the online MATLAB Function Reference provide more detailed descriptions of each property.

Table 1-27: Properties of All Graphics Objects

Property	Description
`BusyAction`	Control events that potentially interrupt executing callback routines.
`Children`	Enhanced behavior allows reordering of child objects.
`CreateFcn`	A callback routine that executes when MATLAB creates a new instance of the specific type of graphics object.
`DeleteFcn`	A callback routine that executes when MATLAB deletes the graphics object.
`HandleVisibility`	Control scope of handle visibility.
`Interruptible`	Now on by default.
`Parent`	Enhanced behavior allows reparenting of graphics objects.
`Selected`	Indicate whether graphics object is in selected state.
`SelectionHighlight`	Determine if graphics objects provide visual indication of selected state.
`Tag`	User-specified object label.

Table 1-28: Axes Properties

Property	Description
`AmbientLightColor`	Color of the surrounding light illuminating all axes child objects when a light object is present.
`CameraPosition`	Location of the point from which the axes is viewed.
`CameraPositionMode`	Automatic or manual camera positioning.
`CameraTarget`	Point in axes viewed from camera position.
`CameraTargetMode`	Automatic or manual camera target selection.
`CameraUpVector`	Determine camera rotation around the viewing axis.
`CameraUpVectorMode`	Default or user-specified camera orientation.
`CameraViewAngle`	Angle determining the camera field of view.
`CameraViewAngleMode`	Automatic or manual camera field of view selection.
`DataAspectRatio`	Relative scaling of *x*-, *y*-, and *z*-axis data units.
`DataAspectRatioMode`	Automatic or manual axis data scaling.
`FontUnits`	Units used to interpret the `FontSize` property (allowing normalized text size).
`Layer`	Draw axis lines below or above child objects.
`NextPlot`	Enhanced behavior supports `add`, `replace`, and `replacechildren` options.
`PlotBoxAspectRatio`	Relative scaling of axes plot box.

Table 1-28: Axes Properties (Continued)

Property	Description
`PlotBoxAspectRatioMode`	Automatic or manual selection of plot box scaling.
`Projection`	Select orthographic or perspective projection type.
`TickDirMode`	Automatic or manual selection of tick mark direction (allowing you to change view and preserve the specified `TickDir`).
`XAxisLocation`	Locate *x*-axis at bottom or top of plot.
`YAxisLocation`	Locate *y*-axis at left or right side of plot.

Table 1-29: Figure Properties

Property	Description
`CloseRequestFcn`	Callback routine executed when you issue a `close` command on a figure.
`Dithermap`	Colormap used for truecolor data on pseudocolor displays.
`DithermapMode`	Automatic dithermap generation.
`IntegerHandle`	Integer or floating-point figure handle.
`PaperPositionMode`	WYSIWYG printing of figure.
`NextPlot`	Enhanced behavior supports `add`, `replace`, and `replacechildren` options.
`PointerShapeCData`	User-defined pointer data.
`PointerShapeHotSpot`	Active point in custom pointer.
`Renderer`	Select painters or Z-buffer rendering.

Table 1-29: Figure Properties (Continued)

Property	Description
RendererMode	Enable MATLAB to select best renderer automatically.
Resize	Determine if Figure window is resizeable.
ResizeFcn	Callback routine executed when you resize the Figure window.

Table 1-30: Image Properties

Property	Description
CData	Enhanced behavior allows truecolor (RGB values) specification.
CDataMapping	Select direct or scaled interpretation of indexed colors.
EraseMode	Control drawing and erasing of image objects.

Table 1-31: Light Properties

Property	Description
Color	Color of the light source.
Position	Place the light source within axes space.
Style	Select infinite or local light source.

Table 1-32: Line Properties

Property	Description
`Marker`	The marker symbol to use at data points (markers are now separate from line style).
`MarkerEdgeColor`	The color of the edge of the marker symbol.
`MarkerFaceColor`	The color of the face of filled markers.

Table 1-33: Patch Properties

Property	Description
`AmbientStrength`	The strength of the axes ambient light on the particular patch object.
`CData`	Enhanced behavior allows truecolor (RGB values) specification.
`CDataMapping`	Select direct or scaled interpretation of indexed colors.
`DiffuseStrength`	Strength of the reflection of diffuse light from light objects.
`FaceLightingAlgorithm`	Lighting algorithm used for patch faces.
`Faces`	The vertices connected to define each face.
`FaceVertexCData`	Color specification when using the `Faces` and `Vertices` properties to define a patch.
`LineStyle`	Type of line used for edges.
`Marker`	Symbol used at vertices.
`MarkerEdgeColor`	The color of the edge of the marker symbol.
`MarkerFaceColor`	The color of the face of filled markers.
`MarkerSize`	Size of the marker.

Table 1-33: Patch Properties (Continued)

Property	Description
NormalMode	MATLAB generated or user-specified normal vectors.
SpecularColorReflectance	Control the color of the specularly reflected light from light objects.
SpecularExponent	Control the shininess of the patch object.
SpecularStrength	Strength of the reflection of specular light from light objects.
VertexNormals	Definition of the patch's normal vectors.
Vertices	The coordinates of the vertices defining the patch.

Table 1-34: Root Properties

Property	Description
CallbackObject	Handle of object whose callback is currently executing.
ErrorMessage	Text of the last error message issued by MATLAB.
ErrorType	The type of the error that last occurred.
ShowHiddenHandles	Show or hide graphics object handles that are marked as hidden.
TerminalHideGraphCommand	Command to hide graphics window when switching to command mode.
TerminalDimensions	Size of graphics terminal.

Table 1-34: Root Properties

Property	Description
`TerminalShowGraphCommand`	Command to expose graphics window when switching from command mode to graphics mode.

Table 1-35: Surface Properties

Property	Description
`AmbientStrength`	The strength of the axes ambient light on the particular surface object.
`CData`	Enhanced behavior allows truecolor (RGB values) specification.
`CDataMapping`	Selects direct or scaled interpretation of indexed colors.
`DiffuseStrength`	Strength of the reflection of diffuse light from light objects.
`FaceLightingAlgorithm`	Lighting algorithm used for surface faces.
`Marker`	Symbol used at vertices.
`MarkerEdgeColor`	The color of the edge of the marker symbol.
`MarkerFaceColor`	The color of the face of filled markers.
`MarkerSize`	Size of the marker.
`NormalMode`	MATLAB generated or user-specified normal vectors.
`SpecularColorReflectance`	Control the color of the specularly reflected light from light objects.
`SpecularExponent`	Control the shininess of the surface object.

Table 1-35: Surface Properties (Continued)

Property	Description
SpecularStrength	Strength of the reflection of specular light from light objects.
VertexNormals	Definition of the surface's normal vectors.
Vertices	The coordinates of the vertices defining the surface.

Table 1-36: Text Properties

Property	Description
FontUnits	Select the units used to interpret the FontSize property (allowing normalized text size).
Interpreter	Allow MATLAB to interpret certain characters as TeX commands.

Table 1-37: Uicontrol Properties

Property	Description
Enable	Enable or disable (gray out) uicontrols.
FontAngle	Select character slant.
FontName	Select font family.
FontSize	Select font size.
FontUnits	Select the units used to interpret the FontSize property (allowing normalized text size).
FontWeight	Select the weight of text characters.

Table 1-37: Uicontrol Properties (Continued)

Property	Description
`ListboxTop`	Select the listbox item to display at the top of the listbox.
`SliderStep`	Select the size of the slider step.
`Style`	Enhanced to include listbox device.

Table 1-38: Uimenu Properties

Property	Description
`Enable`	Enable or disable (gray out) uicontrols.

Improvements to Graphical User Interfaces (GUIs)

General GUI Enhancements

MATLAB 5.0 provided general enhancements that are useful in the GUI area:

- Starting MATLAB with the –`nosplash` argument suppresses the splash screen on UNIX.
- Using the `CloseRequestFcn` callback can abort a figure `close` command.
- Stacking of figure and axes graphics objects can be varied to affect the order in which MATLAB displays these objects.
- The mouse pointer can be set to a number of different symbols or you can create a custom figure pointer.
- On the Windows platforms, edit controls now have a three-dimensional appearance.

MATLAB 5.0 provided features that make it easier to create MATLAB GUIs. Major enhancements includedst box objects to display and select one or more list items. You can also create modal or non-modal error, help, and warning message boxes. In addition, uicontrol edit boxes now support multiline text.

Table 1-39: New GUI Functions

Function	Description
`msgbox`	Display message box.
`dragrect`	Drag pre-defined rectangles.
`inputdlg`	Display a dialog box to input data.
`questdlg`	Question dialog.
`rbbox`	Rubberband box.
`selectmoveresize`	Interactively select, move, or resize objects.

MATLAB 5.0 also added more flexibility in callback routines. You can specify callbacks that execute after creating, changing, and deleting an object.

Table 1-40: New Program Execution Functions

Function	Description
`uiresume`	Resume suspended M-file execution.
`uiwait`	Block program execution.
`waitfor`	Block execution until a condition is satisfied.

Guide

Guide is a Graphical User Interface (GUI) design tool. The individual pieces of the Guide environment are designed to work together, but they can also be used individually. For example, there is a Property Editor (invoked by the command `propedit`) that allows you to modify any property of any Handle Graphics object, from a figure to a line. Point the Property Editor at a line and you can change its color, position, thickness, or any other line property.

The Control Panel is the centerpiece of the Guide suite of tools. It lets you "control" a figure so that it can be easily modified by clicking and dragging. As an example, you might want to move a button from one part of a figure to another. From the Control Panel you put the button's figure into an editable state, and then it's simply a matter of dragging the button into the new position. Once a figure is editable, you can also add new uicontrols, uimenus, and plotting axes.

Table 1-41: Guide Tools

Tool	Command	Description
Control Panel	`guide`	Control figure editing.
Property Editor	`propedit`	Modify object properties.
Callback Editor	`cbedit`	Modify object callbacks.
Alignment Tool	`align`	Align objects.
Menu Editor	`menuedit`	Modify figure menus.

Enhanced Application Program Interface (API)

The MATLAB 5.0 API introduced data types and functions not present in MATLAB 4. This section summarizes the important changes in the API. For details on any of these topics, see the *MATLAB Application Program Interface Guide*.

New Fundamental Data Type

The MATLAB 4 `Matrix` data type is obsolete. MATLAB 5.0 programs use the `mxArray` data type in place of `Matrix`. The `mxArray` data type has extra fields to handle the richer data constructs of MATLAB 5.0.

Functions that expected `Matrix` arguments in MATLAB 4 expect `mxArray` arguments in MATLAB 5.0.

New Functions

The API introduced many new functions that work with the C language to support MATLAB 5.0 features.

Support for Structures and Cells

MATLAB 5.0 introduced structure arrays and cell arrays. Therefore, the MATLAB 5.0 API introduced a broad range of functions to create structures and cells, as well as functions to populate and analyze them. See Chapter 2 for a complete listing of these functions.

Support for Multidimensional Arrays

The MATLAB 4 `Matrix` data type assumed that all matrices were two-dimensional. The MATLAB 5.0 `mxArray` data type supports arrays of two or more dimensions. The MATLAB 5.0 API provides two different `mxCreate` functions that create either a two-dimensional or a multidimensional `mxArray`.

In addition, MATLAB 5.0 introduced several functions to get and set the number and length of each dimension in a multidimensional `mxArray`.

Support for Nondouble Precision Data

The MATLAB 4 `Matrix` data type represented all numerical data as double-precision floating-point numbers. The MATLAB 5.0 `mxArray` data type can store numerical data in six different integer formats and two different floating-point formats.

Note: Although the MATLAB API supports these different data representations, MATLAB itself does not currently provide any operations or functions that work with nondouble-precision data. Nondouble precision-data may be viewed, however.

Enhanced Debugging Support

MATLAB 5.0 included more powerful tools for debugging C MEX-files. The `–argcheck` option to the `mex` script provides protection against accidental misuse of API functions (such as passing `NULL` pointers). In addition, there is increased documentation on troubleshooting common problems.

Enhanced Compile Mechanism

MATLAB 5.0 replaced the old `cmex` and `fmex` scripts with `mex`, which will compile C or Fortran MEX-files. All compiler-specific information was moved to easily readable and highly configurable options files. The `mex` script has a configurable set of flags across platforms and can be accessed from within MATLAB via the `mex.m` M-file.

MATLAB 4 Feature Unsupported in MATLAB 5.0

Non-ANSI C Compilers

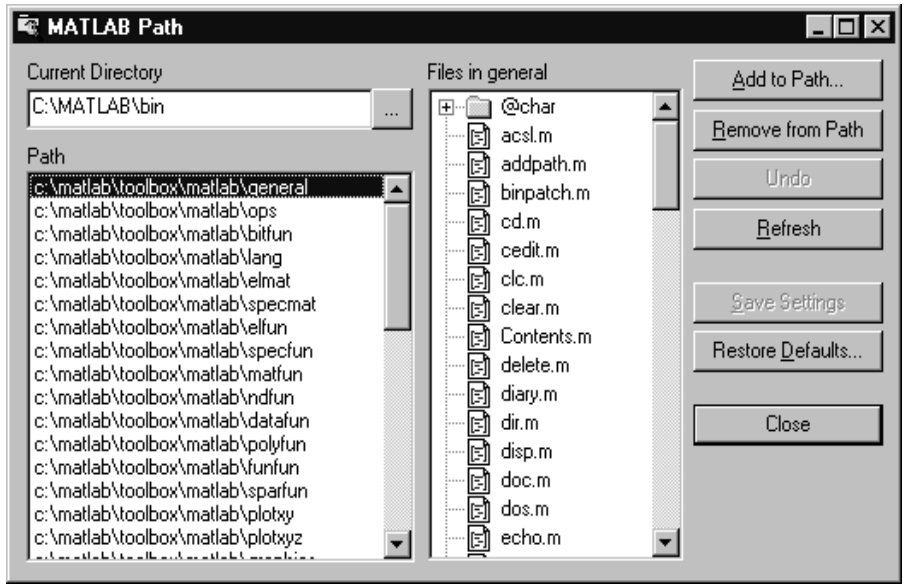
MATLAB 4 let you compile MATLAB applications with non-ANSI C compilers. MATLAB 5.0 required an ANSI C compiler.

New Platform-Specific Features

Microsoft Windows

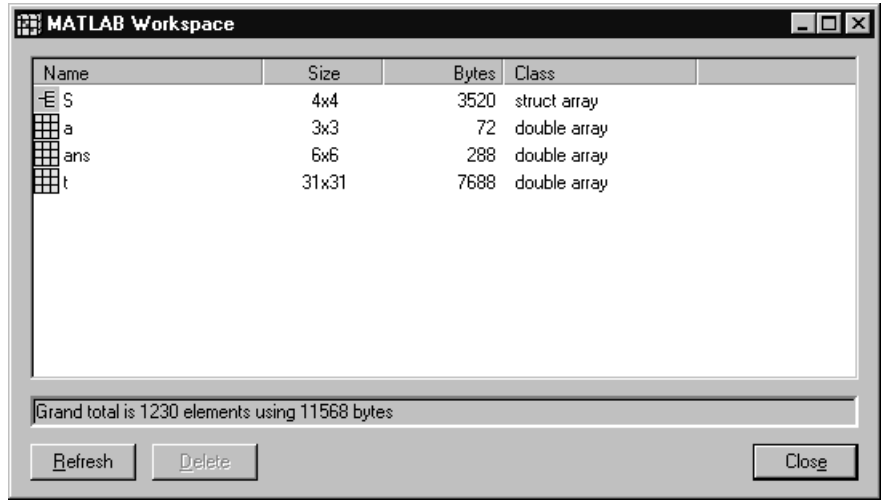
Path Browser

The Path Browser lets you view and modify the MATLAB search path. All changes take effect in MATLAB immediately.



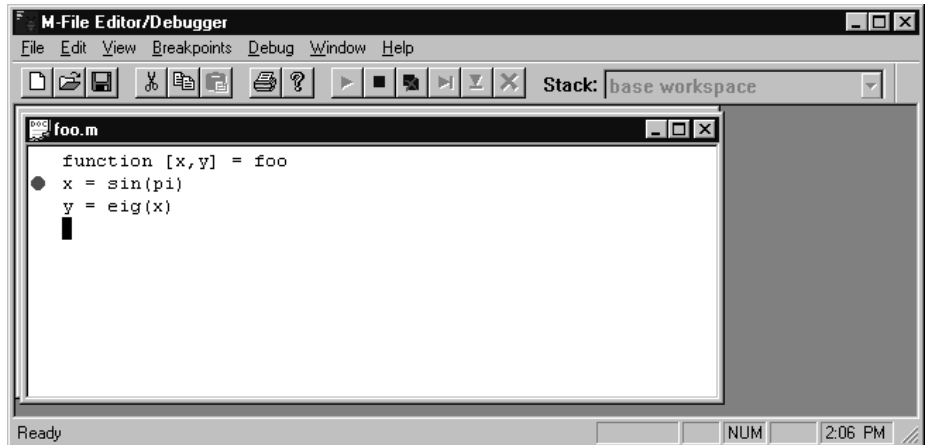
Workspace Browser

The Workspace Browser lets you view the contents of the current MATLAB workspace. It provides a graphical representation of the traditional `whos` output. In addition, you can clear workspace variables and rename them.



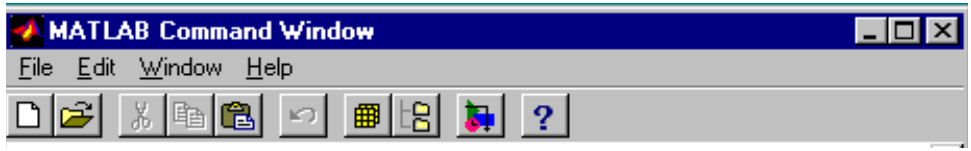
M-File Editor/Debugger

The graphical M-file editor/debugger allows you to set breakpoints and single-step through M-code. The M-file editor/debugger starts automatically when a breakpoint is hit. When MATLAB is installed, this program becomes the default editor.



Command Window Toolbar

There is now a toolbar for the Command Window (you can choose whether or not to display it). The toolbar provides single-click access to several commonly used operations.:



Using the toolbar icons, from left to right, you can:

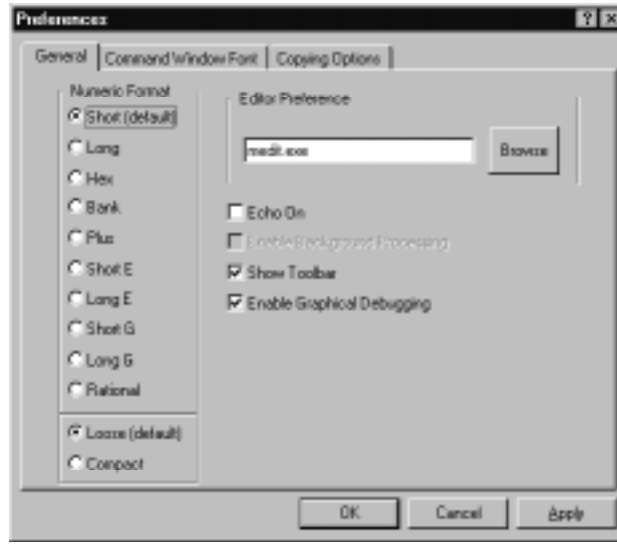
- Open a new editor window
- Open a file for editing
- Cut, copy, paste, and undo (standard Windows icons)
- Open the Workspace Browser
- Open the Path Browser

- Create new Simulink model (if Simulink is installed)
- Access the Help facility

New Dialog Boxes

New **Preferences** dialog boxes are accessible through the **File** menu. Some of these were previously available through the **Options** menu in MATLAB 4. There are three categories of preferences:

- General
- Command Window Font
- Copying Options



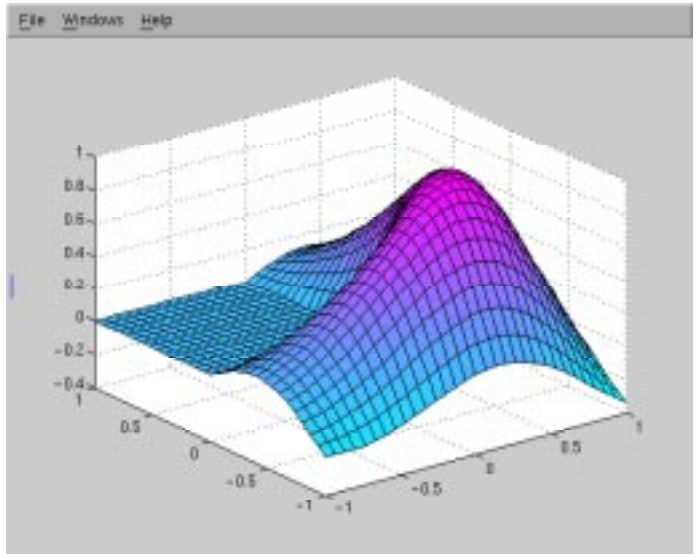
16-bit Stereo Sound

MATLAB 5.0 supports 16-bit stereo sound on the Windows platform.

UNIX Workstations

Figure Window Toolbar

The MATLAB 5.0 Figure window provided a toolbar with a **File** pull-down menu. Selecting the **Print** option on the **File** menu activates a set of push buttons that allows easy setting of the most frequently used print options.

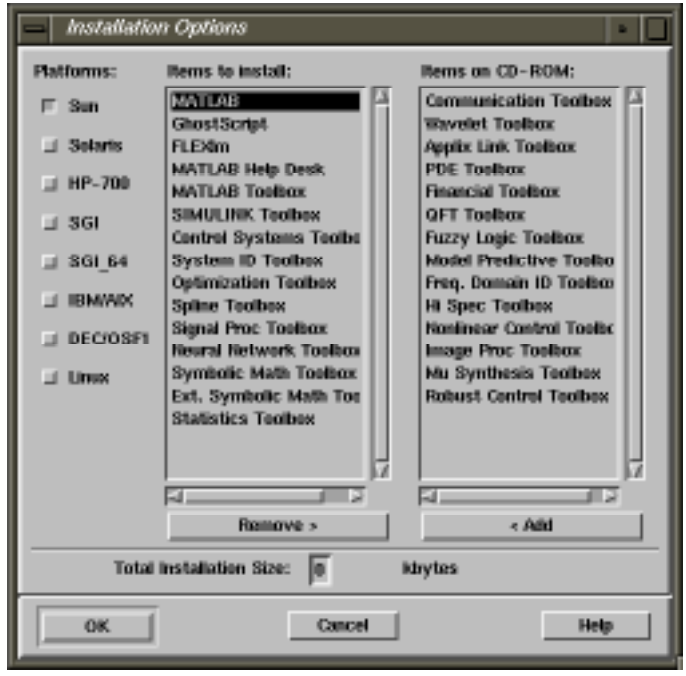


Path Editor

The `pathedit` command displays a GUI that allows you to view and modify your MATLAB search path.

Simplified Installation Procedure

The MATLAB 5.0 installation procedure uses a GUI to select or deselect products and platforms.



2

Upgrading to MATLAB 5.0

Upgrading from MATLAB 4 to MATLAB 5.0

MATLAB 5.0 was a major upgrade to MATLAB 4. Although The MathWorks endeavors to maintain full upwards compatibility between subsequent releases of MATLAB, inevitably there are situations where this is not possible. In the case of MATLAB 5.0, there are a number of changes that you need to know about in order to migrate your code from MATLAB 4 to MATLAB 5.0.

It is useful to introduce two terms in discussing this migration. The first step in converting your code to MATLAB 5.0 is to make it MATLAB 5.0 *compatible*. This involves a rather short list of possible changes that let your M-files run under MATLAB 5.0. The second step is to make it MATLAB 5.0 *compliant*. This means making further changes so that your M-file is not using obsolete, but temporarily supported, features of MATLAB. It also can mean taking advantage of MATLAB 5.0 features like the new data constructs, graphics, and so on.

There are a relatively small number of things that are likely to be in your code that you will have to change to make your M-files MATLAB 5.0 compatible. Most of these are in the graphics area.

There are a somewhat larger number of things you can do (but don't have to) to make your M-files fully MATLAB 5.0 compliant. To help you gradually make your code compliant, MATLAB 5.0 displays warning messages when you use functions that are obsolete, even though they still work correctly.

Note: The changes described here all apply to upgrading from MATLAB 4 to MATLAB 5.0. If you are upgrading from MATLAB 4 to Release 11 (MATLAB 5.3), you should also read the *Release 11 New Features* document.

Converting M-Files to MATLAB 5.0

This section describes some changes you can make to your code to eliminate error messages and warnings due to incompatible and noncompliant statements.

Table 2-1: Language Changes

Function	Change	Action
`auread`, `auwrite`	New syntax.	Change call to use new syntax.
`bessel` functions	The `bessel` functions no longer produce a table for vector arguments of the same orientation.	For example, in `besselj(nu,x)`, specify `nu` as a row and `x` as a column to produce a table.
`case`, `otherwise`, `switch`	`case`, `otherwise`, and `switch` cannot be used as variable names.	Rename your variables.
`dialog`	`dialog.m` now creates a modal dialog.	Use the `msgbox` function instead.
`echo`	`echo` does not display multiline matrices.	Update code.
`end`	extra `end` statements.	Remove redundant `end` statements.
`eps`	`eps` is a function. `eps = 0` no longer redefines `eps` for other functions (it makes a local variable called `eps` in the current workspace). Functions that base their tolerance on an externally defined `eps` won't work.	Change code accordingly.

Table 2-1: Language Changes (Continued)

Function	Change	Action

`for`	`for` loop variable different after loop for empty loops. In MATLAB 4:
	`i = 10;`
	`for i = 1:0, %goes nowhere`
	`end`
	`i`
	produces `i = 10`.
	In MATLAB 5.0 it produces `i = []`.
		Protect the `for` loop with an `isempty` call:
		`i = 10;`
		`if ~isempty(n)`
		`  for i=1:n`
		`  end`
		`end`
		`i`

`global`	Undefined globals	Define globals before they are used. Always put the `global` statement at the top of the M-file (just below the help comments). MATLAB 4 produced a link in the workspace to an uninitialized global. It shows up in `whos` but `exist` returns `0`. Do not use `exist` to test for the first time the global has been accessed. Use `isempty`.
`gradient`	`gradient` no longer produces complex output.	Use two outputs in the 2-D case.
`input`	`input('prompt','s')` no longer outputs an initial line feed. Prompts now show up on the same line.	Update code accordingly if this causes a display problem. Add `\n` in the prompt string to force a line feed.

Table 2-1: Language Changes (Continued)

Function	Change	Action
`interp1`	The old `interp1` syntax (`interp1(x,n)`) no longer calls `interpft`. A warning was in place in MATLAB 4.	Update code accordingly.
	`interp1` now returns a row vector when given a row vector. It used to return a column vector.	Transpose the output of `interp1` to produce the MATLAB 4 result when `xi` is a row vector.
	`interp1('spline')` returns `NaN`'s for out of range values.	Use `spline` directly.
`interp2`	The old `interp2` syntax (`interp2(x,y,xi)`) no longer calls `interp1`. A warning was in place in MATLAB 4.	Update code accordingly.
`interp3`	The old `interp3` syntax (`interp3(z,m,n)` or `interp3(x,y,z,xi,yi)`) no longer calls `griddata`. A warning was in place in MATLAB 4. `interp3` is now 3-D interpolation.	Update code accordingly.
Automeshing interpolation commands	Interpolation automeshing has changed. `griddata`, `interp2`, `interp3`, `interpn`, and `bessel*` now automesh if `(xi,yi)` or `(nu,z)` are vectors of different orientations. Previously they automeshed if the vectors were different size.	When `xi` and `yi` are vectors of the same orientation but different lengths, change calls such as `interp2(...,xi,yi)` to `interp2(...,xi,yi')`.
`isempty`	`A == []` and `A ~= []` as a check for an empty matrix produce warning messages.	Use `isempty(A)` or `~isempty(A)`. In a future version `A == []` will produce an empty result.

Table 2-1: Language Changes (Continued)

Function	Change	Action
`isspace`	`isspace` only returns `true` (1) on strings. `isspace(32)` is 0 (it was 1 in MATLAB 4).	Wrap your calls to `isspace` with `char`.
`logical`	Some masking operations where the mask isn't defined using a logical expression now produce an out of range index error.	Wrap the subscript with a call to `logical` or use the logical expression `A~=0` to produce MATLAB 4 behavior.
	Boolean indexing is no longer directly supported.	Use `logical` to create the index array.
`matlabrc`	On the PC, MATLAB no longer stores the path in `matlabrc`.	MATLAB for PC and UNIX now uses `pathdef.m`.
`max`	`max(size(v))`, as a means to determine the number of elements in a vector `v`, fails when `v` is empty. `max` ignores `NaN`s.	Use `length(v)` in place of `max(size(v))`.
`min`	`min` ignores `NaN`s.	Change code if necessary.
`nargin`, `nargout`	`nargin` and `nargout` are functions.	`nargout = nargout–1` (and any similar construction) is an error. To work around this change, assign `nargin` to a local variable and increment that variable. Rename all occurrences of `nargin` to the new variable. The same holds true for all functions.

Table 2-1: Language Changes (Continued)

Function	Change	Action
ones	A(ones(size(A))) no longer produces A.	This statement produces copies of the first element of A. Use A(ones(size(A))~=0) or just A to produce the MATLAB 4 behavior.
	No longer accepts column vector.	Size vector must be a row vector with integer elements.
	Functions such as ones, eye, rand, and zeros give an error if supplied with a matrix argument (such as zeros(A)).	Use the syntax ones(size(A)) instead.
polyfit	Second output now a structure.	Change code to access structure component.
print	–ocmyk is now –cmyk.	Update code accordingly.
	–psdefcset is now –adobecset.	Update code accordingly.
	GIF format no longer supported.	Use alternate format.
	Texture mapped surfaces do not print with painter's algorithm.	Use –zbuffer.
rand	rand('normal') and rand('uniform') no longer supported.	Use randn for normally distributed and rand for uniformly distributed random numbers.
round	Subscripts must be integers.	To reproduce MATLAB 4 behavior, wrap noninteger subscripts with round(). Strings are no longer valid subscripts (since they are not integers in the strict sense).
slice	slice no longer requires the number of columns (ncols) argument.	Update code accordingly.

Table 2-1: Language Changes (Continued)

Function	Change	Action
`sound`	Doesn't autoscale.	Use `soundsc`.
`strcmp` `strncmp`	`strcmp` and `strncmp` now return `false` (0) when any argument is numeric. They used to perform an `isequal`.	Call `isequal` for all nonstrings you want to compare.
`wavread`, `wavwrite`	New syntax.	Change call to use new syntax.
`zeros`	No longer accepts column vector.	Size vector must be a row vector with integer elements.
Note: The following language changes do not directly apply to specific functions.		
	`a(:) = b` where `a` doesn't exist creates an error. This used to do the same thing as `a = b(:)` when `a` didn't exist.	Either initialize `a` or use `a = b(:)` instead.
	Must use an explicitly empty matrix to delete elements of an array, as in `a(i) = []` or `a(i,:) = []`. This syntax works for all built-in data types (including cell arrays and structures).	Change code accordingly.
	The syntax `a(i) = B`, when `B` is empty, no longer deletes elements.	Use `a(i) = []` instead.
	An attempt to delete elements of an array outside its range is no longer (incorrectly) ignored. An error is generated.	Change code accordingly.

Table 2-1: Language Changes (Continued)

Function	Change	Action
	Undefined variables.	To reproduce MATLAB 4 behavior, initialize your variable to the empty matrix ([ ]) or empty string (' ').
	Undefined outputs.	To reproduce MATLAB 4 behavior, initialize your outputs to the empty matrix ([ ]).
	Indices must be integers. Strings are no longer valid indices.	Use `a(round(ind))` to get MATLAB 4 behavior.
	_,^,{, and } are now interpreted, not displayed.	Use _,\^, \{, and\}.
	Concatenating a string and a double truncates the double.	Use `double` to convert the string before concatenating.
	Input arguments are no longer evaluated left to right.	Evaluate input arguments before passing them to a function.

	String handling difference.
	In MATLAB 4
	`a = 32*ones(1,10);`
	`a(1:5) = 'hello'` produces `'hello'`.
	In MATLAB 5.1, it produces:
	`104 101 108 11 32 32 32 32 32.`
		Initialize `a` to be a character array or convert it after assignment.

	Using inline matrix constants and continued matrix constants inside function calls:
	`fun(arg1,[1 2 3`
	`3 4 5,`
	`5 6 6])`
	is a syntax error.
		Put continuation dots and semicolon after each matrix line.

Table 2-1: Obsolete Language Functions

Obsolete Function	Action
`casesen`	Remove the call.
`csvread`, `csvwrite`	Use `dlmread(`*`filename`*`,',')` and `dlmwrite(`*`filename`*`,',')`.
`ellipk`	Replace with `ellipke`.
`extent`	Replaced by Extent property.
`figflag`	Use `findobj`.
`finite`	Rename to `isfinite`. `finite` was removed in Release 11 (MATLAB 5.3).
`fwhich`	Use `which`.
`hthelp`	`hthelp` works in Release 11 (MATLAB 5.3), but will not be further developed or supported. Use `helpwin`.
`htpp`	Use `helpwin`.
`inquire`	Use `set` and `get` to obtain the current state of an object or of MATLAB.
`inverf`	Rename to `erfinv`.
`isdir`	Use `dir`.
`layout`	No replacement in Release 11 (MATLAB 5.3).
`loadhtml`	Use `helpwin` or `doc`.
`matq2ws`	Replaced by `assignin` and `evalin`.
`matqdlg`	Replaced by `assignin` and `evalin`.
`matqparse`	Replaced by `assignin` and `evalin`.
`matqueue`	Replaced by `assignin` and `evalin`.
`menulabel`	Bug in Handle Graphics is now fixed.
`mexdebug`	Rename to `dbmex`.

Table 2-1: Obsolete Language Functions (Continued)

Obsolete Function	Action
`ode23p`	Use `ode23` with no left-hand arguments or set an output function with `odeset`.
`polyline`, `polymark`	Use the `line` object or `plot`.
`printmenu`	No replacement in Release 11 (MATLAB 5.3).
`saxis`	Use `soundsc`.
`ws2matq`	Replaced by `assignin` and `evalin`.

Table 2-2: Graphics Function Changes

Function	Change	Action
`figure`	In MATLAB 4 if a figure extended past the top of the window, it was adjusted to be visible. MATLAB 5.0 performs no adjustment.	Avoid hardcoded Figure positions.
`get`	`get(h,'currentfigure')` and `get(h,'currentaxes')` no longer create a Figure or an Axes if one doesn't exist. They return `[]` in that case.	`gcf` and `gca` always return a valid handle. Use `gcf` and `gca` instead of the `get` function in this context.
	In MATLAB 4 you could determine if a graphics object had a default value set by passing its handle in a query like `get(gca,'DefaultAxesColor')`.	In MATLAB 5.0 make the query on the object's ancestor, e.g., `get(gcf,'DefaultAxesColor')` or `get(0,'DefaultAxesColor')`

Table 2-2: Graphics Function Changes (Continued)

Function	Change	Action
`plot`	MATLAB 4 plots may have elements that are the wrong color. MATLAB 5.0 defaults to a white background on all platforms. (MATLAB 4 defaulted to black)	Use `colordef` to control your color defaults. Typically, you will put a call to `colordef` in `startup.m`. To get the MATLAB 4 defaults, use `colordef none`.
	`plot` line styles `c1` through `c15` and `i` are no longer supported	Use a 1-by-3 RGB ColorSpec instead. `i` is the same as `get(gca,'color')` or `get(gcf,'color')` when the Axes color is `'none'`.
`rotate`	`rotate alpha` is reversed from MATLAB 4.	If your call looked like `rotate(h,[theta phi],alpha)`, change to `rotate(h,[theta phi],–alpha[0 0 0])`.
`text`	`text(S)` when `S` is a multirow character array formerly produced one handle per row. Now it produces one multiline text handle.	Rewrite code so that it doesn't assume a specific number of handles.

Table 2-2: Graphics Function Changes (Continued)

Function	Change	Action
`uicontrol`	The default Uicontrol text horizontal alignment is centered in MATLAB 5.0. (In MATLAB 4 we used to left align text and ignore the alignment property.)	Explicitly set the horizontal alignment when you create Uicontrol Text objects.
	In MATLAB 4, Uicontrols of style `'edit'` executed their callback routine whenever you moved the pointer out of the edit box. In MATLAB 5.0, edit controls execute their callbacks after you perform a specific action.	The callback is called when: • **Return** key is pressed (single-line edits only) • Focus is moved out of the edit by: ▪Clicking elsewhere in the Figure (on another Uicontrol or on another graphical object) ▪Clicking in another Figure ▪Clicking on the menu bar (X Window systems only)
Note: The following change does not directly apply to a specific function.		
	MATLAB 5.0 sets font size selection to match platform conventions. A MATLAB 4 font selection may be a different size in MATLAB 5.0.	Resize font appropriately.

Table 2-3: Graphics Property Changes

Property	Object Type	Change	Action
`AspectRatio`	Axes	Obsolete	Replace with `DataAspectRatio` and `PlotBoxAspectRatio`.
`BackgroundColor`	Uimenu	Obsolete	Do not use.

Table 2-3: Graphics Property Changes (Continued)

Property	Object Type	Change	Action
`CurrentMenu`	Figure	Becoming obsolete. No warning message produced.	Replace with the function `gcbo`.
`EraseMode`	Image, Line, Patch, Surface, Text	Now use `xor` against the Axes color rather than the Figure color. For non-normal `erasemode`, MATLAB recomputes Axes limits only when you fully update the display (e.g., with a `drawnow` command).	Modify code as appropriate. Set the Axes limits (and other properties you depend upon) before using non-normal modes to create animation.
`ExpFontAngle`	Axes, Text	Obsolete	Do not use.
`ExpFontName`	Axes, Text	Obsolete	Do not use.
`ExpFontSize`	Axes, Text	Obsolete	Do not use.
`ExpFontStrikeThrough`	Axes, Text	Obsolete	Do not use.
`ExpFontUnderline`	Axes, Text	Obsolete	Do not use.
`ExpFontUnits`	Axes, Text	Obsolete	Do not use.
`ExpFontWeight`	Axes, Text	Obsolete	Do not use.
`FontStrikeThrough`	Axes, Text	Obsolete	Do not use.
`FontUnderline`	Axes, Text	Obsolete	Do not use.

Table 2-3: Graphics Property Changes (Continued)

Property	Object Type	Change	Action
`LineStyle`	Line, Patch, Surface	Setting the `LineStyle` property to a marker value (such as `'+'`) now produces a warning. Setting the marker style of a line now affects the `Marker` property instead of the `LineStyle` property. Although you will be able to set a line marker using the `LineStyle` property (with a warning), you will not be able to get marker style information from `LineStyle`.	Set the `Marker` property instead. Note that `plot` will continue to take line-color marker line styles. If your code relies on markers in the `LineStyle`, you'll have to change it to use the `Marker` instead.
`NextPlot`	Axes, Figure	Use of value `'new'` is obsolete. Produces warning message.	Use `HandleVisibility` to protect user interfaces from command line users.
`PaletteMode`	Image, Patch, Surface	Renamed	Use `CDataMapping`
`RenderLimits`	Axes	Obsolete	Do not use. Limits are now always accurate.
`SelectionType`	`Figure`	Right mouse button went from Extended in MATLAB 4 to Alternate in MATLAB 5.0.	None required.

Table 2-3: Graphics Property Changes (Continued)

Property	Object Type	Change	Action
`Units`	Axes, Figure, Text, Uicontrol	Units/Position is always order dependent for all objects. In MATLAB 4, it was inconsistent.	The Units property should precede any properties that depend upon it. A command such as `axes('position',[100 200 300 100],'units','pixels')` is not the same as `axes('units','pixels','position',[100 200 300 100])`. In the first case the numbers are interpreted in normalized coordinates.
`WindowID`	Figure	Possibly becoming obsolete.	May be removed in a future release.
`XLim`, `XTick`	Axes	Values must be monotonically increasing.	Sort the ticks: `set(gca,'xtick',sort([3 2 1]))`
`XTickLabels`	Axes	Renamed	Use `XTickLabel`
`YLim`, `YTick`	Axes	Values must be monotonically increasing.	Sort the ticks: `set(gca,'ytick',sort([3 2 1]))`
`YTickLabels`	Axes	Renamed	Use `YTickLabel`

Table 2-3: Graphics Property Changes (Continued)

Property	Object Type	Change	Action
`ZLim, ZTick`	Axes	Values must be monotonically increasing.	Sort the ticks: `set(gca,'ztick', sort ([3 2 1])`
`ZTickLabels`	Axes	Renamed	`ZTickLabel`

Converting MATLAB 4 External Interface Programs to the MATLAB 5.0 Application Program Interface

MATLAB 4 External Interface programs, including MEX-files, MAT-file programs, and Engine programs may run without any modification on the MATLAB 5.0 Application Program Interface (API), or they may require modification and/or recompilation. The following pages and flowcharts describe how to determine which of these possibilities applies in your situation and how to choose the appropriate conversion technique.

General Considerations

Non-ANSI C Compilers

MATLAB 4 let you compile External Interface programs with non-ANSI C compilers. MATLAB 5.0 API header files include strict prototyping of API functions and require an ANSI C compiler.

MATLAB Character Strings

MATLAB 4 and MATLAB 5.0 represent string data in different ways. MATLAB 4 supported only one data type. All data was represented as double-precision, floating-point numbers, even individual characters in a string array. A numerical array and a character array differed only by how MATLAB displayed these values. MATLAB 5.0 represents each character in a string array as an `mxChar`, a 16-bit unsigned integer data type. If the `mxArray`'s class is `mxCHAR_CLASS`, the API treats each number in the `mxArray` as an element from the current character set. Character sets are platform specific.

External Interface programs that call the API routines `mxGetString()` and `mxCreateString()` to manipulate strings continue to work. All MEX-files that directly manipulate strings must be rewritten. MAT-file and Engine applications that directly manipulate strings need not be rewritten. However, to recompile these applications under MATLAB 5.0, any code that directly manipulates strings must be rewritten. We highly recommend that you use the API string access and creation routines to do this. To help in this endeavor, we have added a new C routine, `mxCreateCharMatrixFromStrings()`, in addition to `mxGetString()` and `mxCreateString()`, to make it easy to create two-dimensional string matrices.

MEX-File Argument Complexification

In MATLAB 4, if one argument to a MEX-function was complex, all arguments were passed as complex. This is not true in MATLAB 5.0. For example, consider a MEX-function, `myeig(A,B,C)`, that calculates eigenvalues of three matrices. In MATLAB 4, if matrix `A` is complex, `B` and `C` are assumed to be complex matrices as well. In this instance, additional memory is allocated for the complex part of `B` and `C`, and initialized with zero values.

MATLAB 5.0 does not allocate this memory for you. If your MEX-file assumes argument complexification, you will have to rewrite your MEX-file. Each argument to a MEX-function needs to be tested with `mxIsComplex()` to guarantee that an argument indeed has a complex component.

Type Imputation by Process of Elimination

In MATLAB 4 the only way to determine if a matrix was a full, nonstring matrix was by a process of elimination. For example, if your code checked a variable and found that it was a full matrix (nonsparse), and was not a string, you could also assume that the variable was double-precision, floating-point and two-dimensional. In MATLAB 5.0, with the addition of several new data types and support for multidimensional data, this assumption is no longer valid. `mxIs*` routines and `mxGetNumberOfDimensions()` have been added to the C interface so that now you can explicitly check arguments for specific data types and shapes. `mxIsDouble()`, which always returned `1` in MATLAB 4, now correctly returns `1` only if the `mxArray` is of type double precision floating point. `mxIsDouble()` is available in C as well as Fortran. `mxGetN()` returns the total number of elements in dimensions 2-n, and therefore works correctly with multidimensional as well as two dimensional arguments.

Version 3.5 MEX-Files

MEX-files generated under MATLAB 4 using the `–v3.5` switch to the `cmex` or `fmex` script are no longer supported and must be rewritten. Refer to both the MATLAB 4 *External Interface Guide* for information on upgrading to the MATLAB 4 syntax and the section later in this document on recoding for MATLAB 5.0 compliance.

Simulink

By design, Simulink S-functions written in C must be recompiled under the latest version of Simulink. The code is source compatible, but the binary must be recompiled. If you attempt to run a Simulink 1.3 C S -function under Simulink 2.1, you get an appropriate warning message. Simulink S-functions written in Fortran do not have this restriction.

Fortran MEX-File Considerations

The MATLAB Fortran API has not changed from MATLAB 4 to MATLAB 5.0. However, Fortran `mex` users on Windows and the Sun4 must recompile their applications, using the `mex` script for programs that call `mxCreateString()` or `mxGetString()`. Only these platforms, which statically link in the Fortran interface, are affected. No recompilation is required on other platforms.

Rebuilding MEX-Files Loaded in Memory

In MATLAB 4, you could execute `cmex` or `fmex` from within MATLAB by using the `!` command or execute in another window. These methods are no longer supported with MATLAB 5.0. However, the M-file that builds MEX-files, `mex.m`, is available on all platforms and can safely rebuild loaded MEX-files by unloading them before proceeding with the build. The interface provided by the M-file is identical to the external `mex` script.

Default MEX-File Optimization

In MATLAB 4 MEX-files by default were built with no optimization flags passed to the compiler. In MATLAB 5.0 the default is to optimize MEX-files. See the *MATLAB Application Program Interface Guide* for more information.

Debugging MEX-Files

The `–debug` switch to `cmex`/`fmex` is no longer supported. Instead, on all platforms, MATLAB 5.0 supports MEX-file debugging from within a MATLAB session. (See the debugging sections of the *MATLAB Application Program Interface Guide* for more details). In addition, the `mex` script has been enhanced with the `–argcheck` switch. That switch provides a way for C `mex` users to generate code to check input and output arguments at runtime and issue appropriate error messages if invalid data is detected.

MAT-File External Applications

MAT-file external applications built under MATLAB 4 will continue to work under MATLAB 5.0. Note that the MAT-file format has changed between MATLAB 4 and MATLAB 5.0. Although MATLAB will be able to read MATLAB 4 MAT-files generated by a stand-alone program, stand-alone programs will not be able to read MAT-files in a MATLAB 5.0 format. You can generate MATLAB 4 MAT-files from MATLAB 5.0 by specifically passing the `–v4` switch to the `save` command.

Windows Considerations

MEX-files are generated under MATLAB 5.0 as 32-bit DLLs. The `cmex` and `fmex` batch files have been superseded by `mex` (a PERL script) and a set of compiler-specific option files. These compilers are fully supported for creating MEX or stand-alone applications through MathWorks-supplied option files:

- Watcom C
- Microsoft Visual C++
- Borland C

Microsoft Fortran, currently the only supported Fortran compiler, is supported for MEX applications only. You will not be able simply to recompile your MATLAB 4 Fortran MEX-file source with this new compiler. See “Creating Fortran MEX-files” in the *MATLAB Application Program Interface Guide* for further information.

NDP Fortran, previously supported under MATLAB 4, is not supported in this release.

16-bit DLL MEX-files are no longer supported and cannot be generated. Existing MATLAB 4 REX MEX-files are usable but cannot be created under MATLAB 5.0.

See “Fortran MEX-file Considerations” above for Windows-specific restrictions on creating and accessing MATLAB character strings.

MATLAB 4 C language Engine binaries will not run with MATLAB 5.0. Programs must be recompiled. MATLAB 5.0 data types are currently not supported on the PC from an Engine program. There is currently no support for Fortran Engine or MAT-file programs.

See the *MATLAB Application Program Interface Guide* for instructions on compiling stand-alone programs in MATLAB 5.0.

UNIX Considerations

The `cmex` and `fmex` Bourne shell scripts for building MEX-files have been superseded by `mex`, also a Bourne shell script, that sources an options file, `mexopts.sh` for all platform compiler-specific information. The options file contains all the pertinent compiler and linker switches for building ANSI C and Fortran MEX-file applications. The `.mexrc.sh` file is no longer supported and must be converted to the new format. The `mexdebug` MATLAB command that allows UNIX users to debug their MEX-files while MATLAB is running has been changed to `dbmex`. The behavior of `dbmex` under MATLAB 5.0 is identical to `mexdebug` under MATLAB 4.

See "Fortran Considerations" above for Sun4-specific restrictions on creating and accessing MATLAB character strings.

You can use your existing UNIX Engine and MAT-file binary files unmodified; no recompilation is necessary. Note that MATLAB 4 Engine programs have no access to new MATLAB 5.0 data types. If you try to invoke `engGetMatrix(ep,`*`my_variable`*`)`, and *`my_variable`* is a cell array, structure array, object, etc., the operation automatically fails.

Conversion

Rebuilding MEX-Files

The simplest strategy for converting C MEX-file programs is to rebuild them with the special `–V4` option of `mex`. This option uses `mex` to define a macro `V4_COMPAT` that supports MATLAB 4 syntax and function calls. Therefore, any ANSI C MEX-file source code that compiled cleanly under MATLAB 4 should compile cleanly with the `–V4` option. The resulting MEX-file should run under MATLAB 5.0 just as it ran under MATLAB 4. For example, given C MEX-file MATLAB 4 source code in file `MyEig.c`, recompiling under UNIX with

```
mex –V4 myeig.c
```

yields a MEX-file that MATLAB 5.0 can execute. It is also possible to use `cmex` and `fmex` for compiling C and Fortran source code, but both of these functions simply call `mex`.

The obvious advantage to the –V4 strategy is that it requires very little work on your part. However, this strategy provides only a temporary solution to the conversion problem; there is no guarantee that future releases of MATLAB will continue to support the –V4 option. If you have the time, recoding for MATLAB 5.0 compliance is a better strategy. See "Recoding C Code for MATLAB 5.0 Compliance" below.

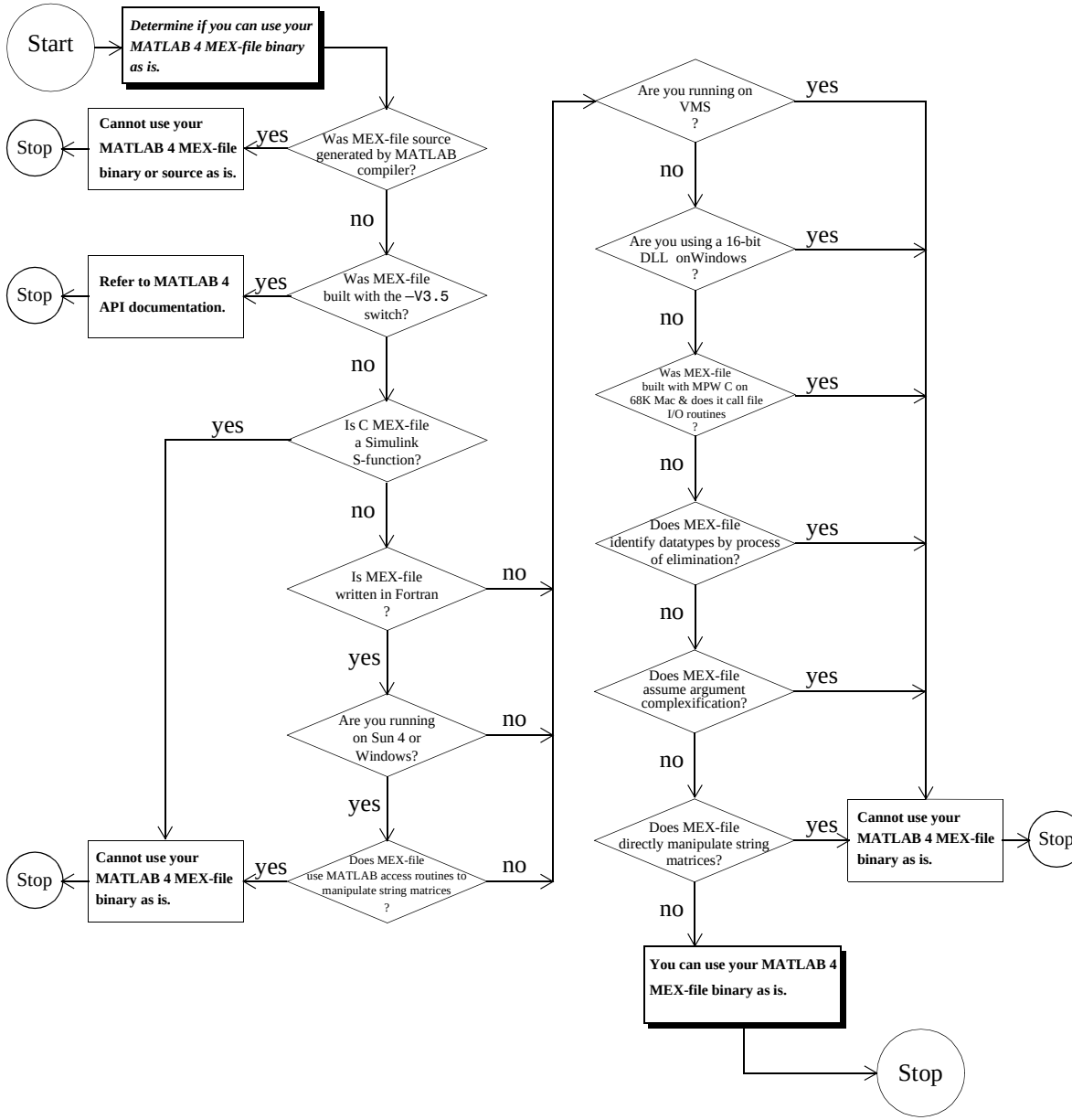
Rebuilding Stand-Alone MAT-File and Engine Programs

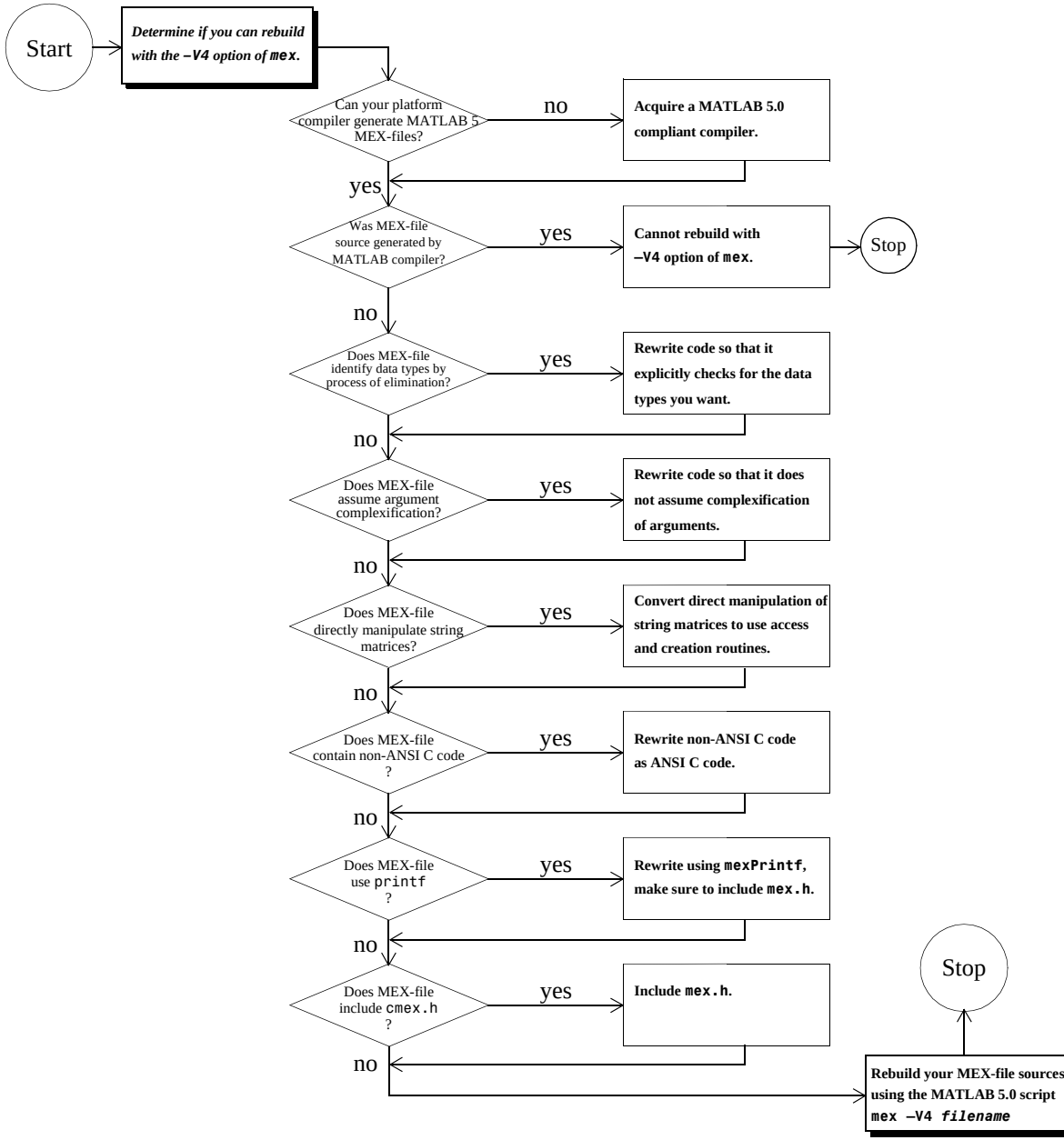
If your source code is ANSI compliant, you can recompile your source without modification by using the compiler flag –DV4_COMPAT. This allows you to avoid recoding, such as rewriting obsolete function calls as their MATLAB 5.0 equivalents. However, the resulting program will have the same restrictions as existing binary files.

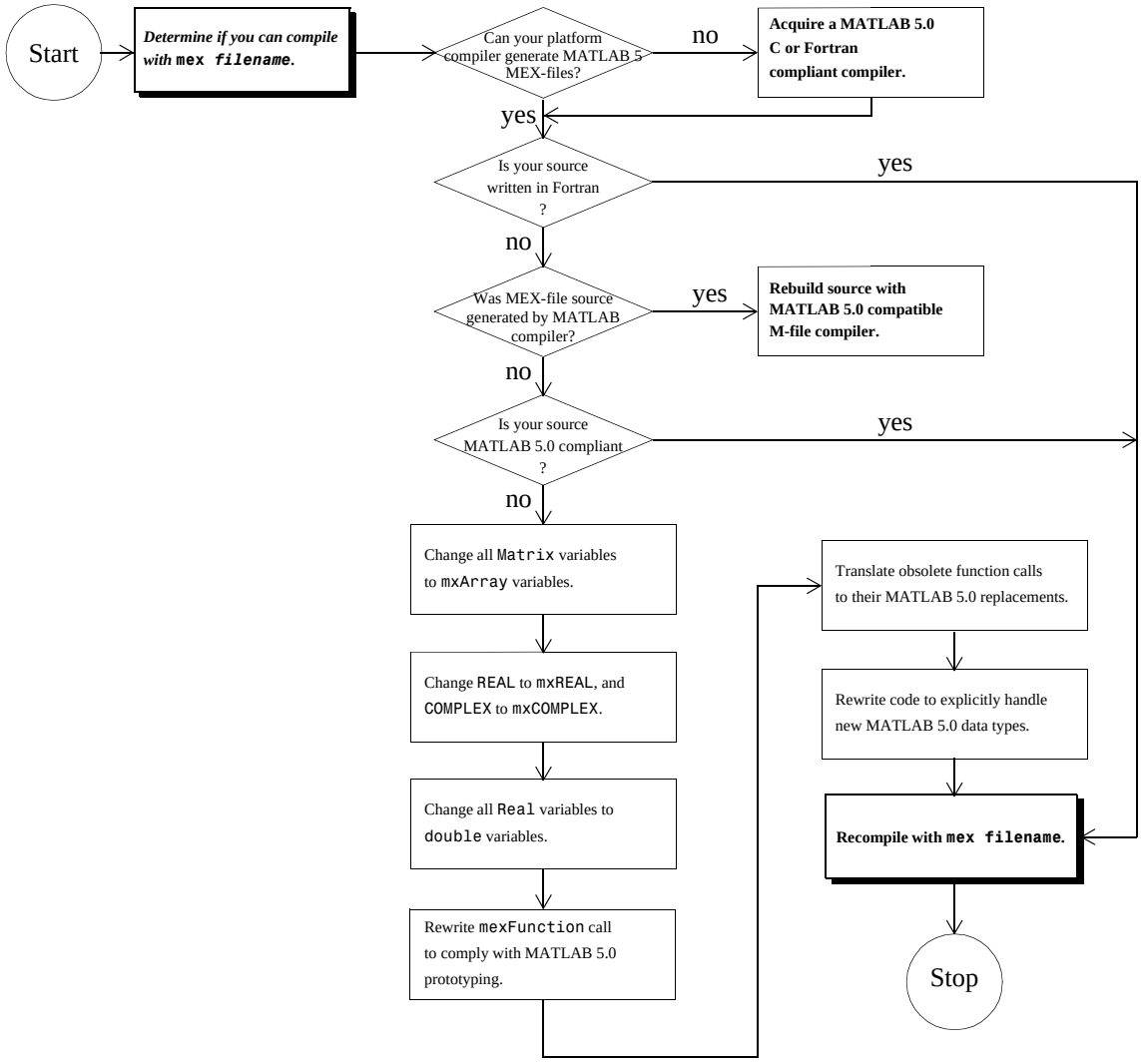
See the *MATLAB Application Program Interface Guide* for instructions on compiling stand-alone programs in MATLAB 5.0.

MEX-File Conversion Flowcharts

The flowcharts below help you determine what steps you should take to run your MATLAB 4 MEX-files under MATLAB 5.0.


Start
Determine if you can use your MATLAB 4 MEX-file binary as is.
Was MEX-file source generated by MATLAB compiler?
yes
Cannot use your MATLAB 4 MEX-file binary or source as is.
Stop
no
Was MEX-file built with the -V3.5 switch?
yes
Refer to MATLAB 4 API documentation.
Stop
no
Is C MEX-file a Simulink S-function?
yes
no
Is MEX-file written in Fortran ?
no
yes
Are you running on Sun 4 or Windows?
no
yes
Does MEX-file use MATLAB access routines to manipulate string matrices ?
yes
Cannot use your MATLAB 4 MEX-file binary as is.
Stop
no
Are you running on VMS ?
yes
no
Are you using a 16-bit DLL onWindows ?
yes
no
Was MEX-file built with MPW C on 68K Mac & does it call file I/O routines ?
yes
no
Does MEX-file identify datatypes by process of elimination?
yes
no
Does MEX-file assume argument complexification?
yes
no
Does MEX-file directly manipulate string matrices?
yes
Cannot use your MATLAB 4 MEX-file binary as is.
Stop
no
You can use your MATLAB 4 MEX-file binary as is.
Stop


Start
Determine if you can rebuild with the –V4 option of mex.
Can your platform compiler generate MATLAB 5 MEX-files?
no
Acquire a MATLAB 5.0 compliant compiler.
yes
Was MEX-file source generated by MATLAB compiler?
yes
Cannot rebuild with –V4 option of mex.
Stop
no
Does MEX-file identify data types by process of elimination?
yes
Rewrite code so that it explicitly checks for the data types you want.
no
Does MEX-file assume argument complexification?
yes
Rewrite code so that it does not assume complexification of arguments.
no
Does MEX-file directly manipulate string matrices?
yes
Convert direct manipulation of string matrices to use access and creation routines.
no
Does MEX-file contain non-ANSI C code ?
yes
Rewrite non-ANSI C code as ANSI C code.
no
Does MEX-file use printf ?
yes
Rewrite using mexPrintf, make sure to include mex.h.
no
Does MEX-file include cmex.h ?
yes
Include mex.h.
no
Stop
Rebuild your MEX-file sources using the MATLAB 5.0 script mex –V4 filename


Start
Determine if you can compile with mex filename.
Can your platform compiler generate MATLAB 5 MEX-files?
no
Acquire a MATLAB 5.0 C or Fortran compliant compiler.
yes
Is your source written in Fortran ?
yes
no
Was MEX-file source generated by MATLAB compiler?
yes
Rebuild source with MATLAB 5.0 compatible M-file compiler.
no
Is your source MATLAB 5.0 compliant ?
yes
no
Change all Matrix variables to mxArray variables.
Change REAL to mxREAL, and COMPLEX to mxCOMPLEX.
Change all Real variables to double variables.
Rewrite mexFunction call to comply with MATLAB 5.0 prototyping.
Translate obsolete function calls to their MATLAB 5.0 replacements.
Rewrite code to explicitly handle new MATLAB 5.0 data types.
Recompile with mex filename.
Stop

Recoding C Code for MATLAB 5.0 Compliance

Recoding your MATLAB 4 C code for MATLAB 5.0 compliance involves:

- Rewriting any non-ANSI C code as ANSI C code. (For details, see an ANSI C book.)
- Changing all `Matrix` variables to `mxArray` variables.

  The MATLAB 4 `Matrix` data type is obsolete; you must change all `Matrix` variables to `mxArray` variables. For example, the `mxCreateSparse` function returns a Matrix pointer in MATLAB 4:

  ```
  Matrix *MySparse;
  MySparse = mxCreateSparse(10,10,110,REAL);
  ```

  To be MATLAB 5.0 compliant, change the code to:

  ```
  mxArray *MySparse;
  MySparse = mxCreateSparse(10,10,110,mxREAL);
  ```

- Rewriting all function prototypes.

  The function prototype of almost every MATLAB 4 function is different in MATLAB 5.0. The two primary prototype changes are

  - All `Matrix` arguments are now `mxArray` arguments.
  - Pointers to read only data are now declared as `const *`.

- For MEX-files, rewriting `mexFunction` to take a constant `mxArray *` as a fourth argument.
- Changing `REAL` to `mxREAL` and `COMPLEX` to `mxCOMPLEX`.

  In any function that requires the specification of real or complex data types, instead of `REAL` and `COMPLEX`, use `mxREAL` and `mxCOMPLEX`. For example, in MATLAB 4 you would write

  ```
  mxCreateSparse(m,n,nzmax,REAL);
  ```

  to create an m-by-n sparse matrix with `nzmax` nonzero real elements. In MATLAB 5.0, the correct syntax for this same function is:

  ```
  mxCreateSparse(m,n,nzmax,mxREAL);
  ```

- Translating obsolete function calls into their MATLAB 5.0 replacements.

  A number of functions have become obsolete. However, MATLAB 5.0 offers replacements for nearly all obsolete functions.

- Handling MATLAB 5.0 new data types.

  You should explicitly check the data type of your input arguments to ensure that you have what you want.

Table 3-5 lists MATLAB 4 External Interface functions along with a description of how to recode those functions to work with MATLAB 5.0.

Table 2-4: Recoding MATLAB 4 Functions for MATLAB 5.0 Compliance

MATLAB 4 Function	MATLAB 5.0 Replacement
`engGetMatrix`	`engGetArray`
`engGetFull`	`engGetArray` followed by calls to the appropriate `mx*` routines
`engPutMatrix`	`engPutArray`
`engPutFull`	`mxCreateDoubleMatrix` followed by `engPutArray`
`engSetEvalCallback`	(Windows platform only) Obsolete in 5.0
`engSetEvalTimeout`	(Windows platform only) Obsolete in 5.0
`engWinInit`	(Windows platform only) Obsolete in 5.0
`matGetMatrix`	`matGetArray`
`matGetNextMatrix`	`matGetNextArray`
`matGetFull`	`matGetArray` followed by calls to the appropriate `mx*` routines
`matPutMatrix`	`matPutArray`
`matPutFull`	`mxCreateDoubleMatrix` followed by `matPutArray`
`mexAtExit`	No change
`mexCallMATLAB`	Second and fourth arguments are `mxArray *`
`mexErrMsgTxt`	No change
`mexEvalString`	No change

Table 2-4: Recoding MATLAB 4 Functions for MATLAB 5.0 Compliance (Continued)

MATLAB 4 Function	MATLAB 5.0 Replacement
`mexFunction`	Second and fourth arguments are `mxArray *` Fourth argument is a `const`
`mexGetEps`	Obsolete; call `mxGetEps` instead

`mexGetFull`	Obsolete; call this sequence instead:
	`mexGetArray(array_ptr, "caller");`
	`name = mxGetName(array_ptr);`
	`m = mxGetM(array_ptr);`
	`n = mxGetM(array_ptr);`
	`pr = mxGetPr(array_ptr);`
	`pi = mxGetPi(array_ptr);`

`mexGetGlobal`	Obsolete; call `mexGetArrayPtr` instead, setting the second argument to `"global"`. Note: it is better programming practice to call `mexGetArray(,"global");`
`mexGetInf`	Obsolete; call `mxGetInf` instead
`mexGetMatrix`	Call `mexGetArray(name,"caller");`
`mexGetMatrixPtr`	Call `mexGetArrayPtr(name,"caller");`
`mexGetNaN`	Obsolete; call `mxGetNaN` instead
`mexIsFinite`	Obsolete; call `mxIsFinite` instead
`mexIsInf`	Obsolete; call `mxIsInf` instead
`mexIsNaN`	Obsolete; call `mxIsNaN` instead
`mexPrintf`	No change

Table 2-4: Recoding MATLAB 4 Functions for MATLAB 5.0 Compliance (Continued)

MATLAB 4 Function	MATLAB 5.0 Replacement

`mexPutFull`	Obsolete; call this sequence instead:
	`mxArray *parray;`
	`int retval;`
	
	`parray = mxCreateDouble(0,0,0);`
	`if(parray == (mxArray*)0) return(1);`
	`mxSetM(parray,m);`
	`mxSetN(parray,n);`
	`mxSetPr(parray,pr);`
	`mxSetPi(parray,pi);`
	`mxSetName(parray,name);`
	
	`retval = mxPutArray(parray,"caller");`
	`mxFree(parray);`
	`return(retval);`

`mexPutMatrix`	Obsolete; call `mexPutArray` instead
`mexSetTrapFlag`	No change
`mxCalloc`	No change
`mxCreateFull`	Obsolete; call `mxCreateDoubleMatrix` instead
`mxCreateSparse`	Returns `mxArray *`
`mxCreateString`	Returns `mxArray *`
`mxFree`	No change
`mxFreeMatrix`	Obsolete; call `mxDestroyArray` instead
`mxGetIr`	First argument is `mxArray *`
`mxGetJc`	First argument is `mxArray *`
`mxGetM`	First argument is `mxArray *`
`mxGetN`	First argument is `mxArray *`

Table 2-4: Recoding MATLAB 4 Functions for MATLAB 5.0 Compliance (Continued)

MATLAB 4 Function	MATLAB 5.0 Replacement
mxGetName	First argument is mxArray *
mxGetNzmax	First argument is mxArray *
mxGetPi	First argument is mxArray *
mxGetPr	First argument is mxArray *
mxGetScalar	First argument is mxArray *
mxGetString	First argument is mxArray *
mxIsComplex	First argument is mxArray *
mxIsDouble	First argument is mxArray * Note that MATLAB 4 stores all data as doubles; MATLAB 5.0 stores data in a variety of integer and real formats.
mxIsFull	Obsolete; call !mxIsSparse instead
mxIsNumeric	First argument is mxArray *
mxIsSparse	First argument is mxArray *
mxIsString	Obsolete; call mxIsChar instead
mxSetIr	First argument is mxArray *
mxSetJc	First argument is mxArray *
mxSetM	First argument is mxArray *
mxSetN	First argument is mxArray *
mxSetName	First argument is mxArray *
mxSetNzmax	First argument is mxArray *
mxSetPi	First argument is mxArray *

Table 2-4: Recoding MATLAB 4 Functions for MATLAB 5.0 Compliance (Continued)

MATLAB 4 Function	MATLAB 5.0 Replacement
`mxSetPr`	First argument is `mxArray *`.
`mxSetString`	Obsolete; MATLAB 5.0 provides no equivalent call since the `mxArray` data type does not contain a string flag. Use `mxCreateCharMatrixFromStrings` to create multidimensional string `mxArrays`.

Index

C

D

E

F

G

H

I

L

M

N

O

P

T

U

V

W

X

Y

Z