MATLAB® C/C++ Graphics Library

The Language of Technical Computing

Computation

Visualization

Programming

The MATH WORKS Inc.

User's Guide

Version 1

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

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

MATLAB C/C++ Graphics Library User's Guide

Printing History: June 1999 New for Version 1.0, Release 11 (Online only)
January 2000 Revised for Version 1.0.2, Release 11 (Online only)

Contents

1 Introduction

2 Creating Stand-Alone Graphics Applications

3 Troubleshooting

4 Reference

1

Introduction

The MATLAB C/C++ Graphics Library

The MATLAB® C/C++ Graphics Library is a collection of MATLAB graphics routines distributed as a single library. The graphics library makes the visualization and GUI-building routines of MATLAB available to stand-alone C and C++ applications. A stand-alone C or C++ application is an executable program that can run independently of the MATLAB interpreted environment. Stand-alone applications are a convenient way to package and distribute a customized MATLAB application.

Using this library, in conjunction with the MATLAB Compiler and the MATLAB C/C++ Math Library, you can create stand-alone applications from M-files that use lines, text, meshes, and polygons as well as interactive graphical user interface components such as menus, push buttons, and dialog boxes.

Note You must use the MATLAB Compiler to create C or C++ stand-alone graphics applications. Calling MATLAB C/C++ Graphics Library routines directly from a C or C++ source module is not supported.

The following sections provide information about:

- "Components of the MATLAB C/C++ Graphics Library" on page 1-2
- "Restrictions" on page 1-3
- "Sources of Additional Information" on page 1-6

After reading these sections, continue with "Installing the MATLAB C/C++ Graphics Library" on page 1-7 and then read "Configuring the MATLAB C/C++ Graphics Library" on page 1-14.

Components of the MATLAB C/C++ Graphics Library

The MATLAB C/C++ Graphics Library contains more than 100 routines, including:

- MATLAB 5.3 built-in graphics functions, such as `surf`, `plot`, `get`, and `set`
- Some commonly used MATLAB 5.3 M-file graphics functions, such as `newplot`, `gcf`, `gca`, `gco`, and `gcbf`

For a complete list of the routines included in the graphics library, see Chapter 4, “Reference.”

Note While only a subset of MATLAB M-file graphics functions are included in the MATLAB C/C++ Graphics Library, the MATLAB Compiler can compile many others graphics M-files, if they are included in a graphics application.

Restrictions

The MATLAB C/C++ Graphics Library supports most MATLAB 5.3 features, including multidimensional arrays, cell arrays, and structures. However, there are some MATLAB features the graphics library does not support, including:

- MATLAB integer types (`int8`, `int16`, `int32`, `uint8`, `uint16`, and `uint32`)
- MATLAB objects
- OpenGL
- `plotedit` command

In addition to these restrictions, the graphics library provides limited support for printing and certain callback coding practices.

Note The graphics library is subject to the same limitations as the MATLAB Compiler and the MATLAB C/C++ Math Library. For example, MATLAB functions that require the MATLAB interpreter, most notably `eval()` and `input()`, are not supported. See the MATLAB Compiler and MATLAB C/C++ Math Library documentation for information about their restrictions.

Graphics Library Printing Support

The graphics library supports printing in stand-alone graphics applications, with some restrictions.

Default Printing. If your application uses default `print` command settings, it should require no modification. When a user of your stand-alone graphics application selects the **Print** option on the figure window **File** menu, the `print` command executed by default sends the current figure to whatever printer the

user has set up as the default printer on their system. In addition, the graphics library honors any figure or axes properties your application sets to customize the printed output.

Custom Printing. If your application uses `print` command switches to specify device drivers and other options, be aware that the graphics library supports only a subset of these switches. For example, the graphics library supports most of the MATLAB built-in drivers, such as the PostScript drivers. However, the graphics library does not support any of the Ghostscript drivers.

Table 1-1 lists the device drivers supported by the graphics library. For more information about specifying device drivers, see "Printing MATLAB Graphics" in *Using MATLAB Graphics*.

Table 1-1: Device Drivers Supported by the Graphics Library

Device	Description
`-dbitmap`	Windows Bitmap (BMP) format (Windows only)
`-deps`	Level 1 black and white Encapsulated PostScript (EPS)
`-depsc`	Level 1 color Encapsulated PostScript (EPS)
`-deps2`	Level 2 black and white Encapsulated PostScript (EPS)
`-depsc2`	Level 2 color Encapsulated PostScript (EPS)
`-dhpgl`	HPGL compatible with HP 7475A plotter
`-dill`	Adobe Illustrator 88 compatible illustration file
`-dps`	Level 1 black and white PostScript
`-dpsc`	Level 1 color PostScript
`-dps2`	Level 2 black and white PostScript
`-dpsc2`	Level 2 color PostScript
`-dwin`	Windows black and white printing services. (Windows only)
`-dwinc`	Windows color printing services. (Windows only)

In addition to device drivers, the MATLAB `print` command supports several command-line options that control various aspects of the print job, such as the renderer used. The graphics library supports the subset of these options that are listed in Table 1-2. For a complete list of print command options, see "Printing MATLAB Graphics" in *Using MATLAB Graphics*.

Table 1-2: print Command Line Options Supported by the Graphics Library

Option	Description
`-adobecset`	Use PostScript default character set encoding
`-append`	Append to existing PostScript file without overwriting
`-cmyk`	Use CMYK colors in PostScript instead of RGB
`-noui`	Suppress printing of user interface controls
`-r`*`number`*	Specify resolution in dots per inch
`-painters`	Render using Painter's algorithm
`-zbuffer`	Render using Z-buffer

Unsupported Application Coding Practices

Certain coding practices, used mainly in callback strings in graphics M-files, that are supported in the MATLAB interpreted environment are not supported in stand-alone graphics applications. For example, MATLAB allows you to specify math expressions in a callback property string. This is not supported in stand-alone graphics applications. See Chapter 3, "Troubleshooting" for more information about unsupported coding practices and how to work around them.

Sources of Additional Information

This manual describes the MATLAB C/C++ Graphics Library but it does not describe how to use MATLAB graphics or specific MATLAB graphics functions. This table lists sources of additional information about these topics and other topics related to using the MATLAB C/C++ Graphics Library. The online reference documentation is available through the MATLAB Help Desk.

Topic	Document
Using MATLAB graphics functions	*Using MATLAB Graphics*
Building graphical user interfaces	*Building GUIs with MATLAB*
Individual MATLAB graphical functions	Online *MATLAB Function Reference*
Using the MATLAB Compiler	*MATLAB Compiler User's Guide*
Using the MATLAB C/C++ Math Library	*MATLAB C Math Library User's Guide* *MATLAB C++ Math Library User's Guide*
Individual MATLAB C and C++ math functions	Online *MATLAB C Math Library Function Reference* Online *MATLAB C++ Math Library Function Reference*

Installing the MATLAB C/C++ Graphics Library

The MATLAB C/C++ Graphics Library is available as a downloadable product from the MathWorks Web site. This section describes:

After reading this section, continue with "Configuring the MATLAB C/C++ Graphics Library" on page 1-14.

Before You Install

Before you can install and use the MATLAB C/C++ Graphics Library, you must have a valid license for this product. When you purchase a MATLAB product, The MathWorks sends you your license in an e-mail message. If you have not received your license, contact The MathWorks immediately via:

- The Web at `www.mathworks.com/mla`. Log in to MATLAB Access using your last name and Access number. MATLAB Access membership is free of charge and available to all customers. The primary contact on each license is automatically enrolled in MATLAB Access and receives an Access number via e-mail from The MathWorks.
- E-mail at `service@mathworks.com`
- Telephone at 508-647-7000, ask for Customer Service
- Fax at 508-647-7001

System Requirements

Software. To use the MATLAB C/C++ Graphics Library to create a stand-alone C or C++ application requires several other MATLAB products:

- MATLAB Version 5.3.1
- MATLAB Compiler Version 2.0.1
- MATLAB C/C++ Math Library Version 2.0.1

You must also have installed on your system an ANSI C or C++ compiler.

Hardware. The MATLAB C/C++ Graphics Library is available for PCs running Microsoft Windows or Linux and for Sun, HP, SGI, and Alpha UNIX platforms. The MATLAB C/C++ Graphics Library is not supported on IBM RS/6000 systems.

For the most up-to-date information about the systems supported by MATLAB, Release 11, see the System Requirements page in the Products area at the MathWorks Web site, `www.mathworks.com`.

Downloading and Installing on PCs

To download and install the graphics library on a PC, follow these instructions.

1. Go to The MathWorks Web site, `www.mathworks.com`, and click on the **Downloads** selection listed under Quick Links.

2. If you have your Personal License Password, click on the **Download Products** link. If you do not have your PLP, follow the appropriate link to obtain it.

3. Enter your name and MATLAB Access number.

4. Select the type of system on which you want to run the graphics library from the list of supported systems, and then click the **Continue** button. You can select multiple systems.

5. Select the MATLAB C/C++ Graphics Library from the list of products you are licensed to install, and then click the **Continue** button.

6. Download the required product archives, and any optional product archives, by right-clicking on their links and selecting the save option provided by your browser. You can specify the folder into which you want to download the product archive and the name you want assigned to the archive.

7. When the download is complete, view the contents of your download folder and double-click on the graphics library archive. This starts the MATLAB installation program. Follow the instructions included in each dialog box displayed by the installation program. You will need your license information to complete the installation.

Note Make sure there is space available in your system's temporary directory before starting the installation program. The name of the temporary directory is defined by the environment variable `%TEMP%`.

For more information about the installation process, read the *Installation Guide for PC*, available in PDF format on the Support page at The MathWorks Web site. Click on the **Documentation** link and go to the **Online Manuals** page to find the documentation in PDF format.

Files Installed on PCs

This table lists the shared libraries (DLLs), include files, and other files installed on a PC as part of a MATLAB C/C++ Graphics Library installation. In the table, `<matlab>` stands for your top-level MATLAB installation directory.

Note On PCs, the MATLAB C/C++ Graphics Library installation includes new versions of several standard MATLAB dynamic link libraries (DLLs).

Table 1-3: List of Files Installed on PCs

Files	Location	Description

`sgl.dll`
`hg_sgl.dll`
`uiw_sgl.dll`
`hardcopy_sgl.dll`
`gui_sgl.dll`
`mpath.dll`
	`<matlab>\bin`	Shared libraries containing stand-alone versions of MATLAB built-in and M-file graphics functions.
		
		All DLLs are in WIN32 format.

`sgl` `sglcpp`	`<matlab>\bin`	MATLAB Compiler bundle files, containing all the compiler options required to build a stand-alone graphics application.

Table 1-3: List of Files Installed on PCs (Continued)

`libsgl.h`
`libsglm.h`
`sgl.def`
	`<matlab>\extern\include`	Graphics library header files (`.h`) and module definition file (`.def`).

`FigureMenuBar.fig` `FigureToolBar.fig`	`<matlab>\extern\include`	Alternate menu bar and toolbar files used with the MATLAB figure window in stand-alone applications.
`flames.m` `flames.mat`	`<matlab>\extern\examples\sgl`	The M-file and MAT file for the graphics library example program.

Verifying Your Installation

To verify that the MATLAB C/C++ Graphics Library has been installed correctly, create a stand-alone application from the example M-file distributed with the library. Chapter 2, "Creating Stand-Alone Graphics Applications" describes how to build a stand-alone application. You can find the graphics library example program in the `<matlab>\extern\examples\sgl` directory.

Downloading and Installing on UNIX Systems

To download and install the MATLAB C/C++ Graphics Library on a UNIX system, follow these instructions:

1. Go to The MathWorks Web site, `www.mathworks.com`, and click on the **Downloads** selection listed under **Quick Links**.
2. If you have your License File, click on the **Download Products** link. If you do not have your License File, follow the appropriate link to obtain it.
3. Enter your name and MATLAB Access number.
4. Select the type of system on which you want to run the graphics library from the list of supported systems, and then click the **Continue** button. You can select multiple systems.
5. Select the MATLAB C/C++ Graphics Library from the list of products you are licensed to install, and then click the **Continue** button.

6 Create a MATLAB installation directory, if it doesn't already exist, in your root directory (for example, `/root/matlab`).

7 Download the required product archives, and any optional product archives, into the MATLAB installation directory by right-clicking their links and selecting the save option provided by your browser. Do not change the names of the archives.

8 In the installation directory, extract files from the `boot.ftp` archive using the UNIX `tar` command

```
tar -xvf boot.ftp
```

This archive contains an installer program that extracts files from the other archives you downloaded.

9 Place a copy of your License File in the MATLAB installation directory and call it `license.dat`.

10 Run the MATLAB installation program and follow the instructions presented on each dialog box.

```
install_matlab -X
```

For more information about the installation process, read the *Installation Guide for UNIX*, available in PDF format on the Support page at The MathWorks Web site. Click on the **Documentation** link and go to the **Online Manuals** page to find the documentation in PDF format.

Files Installed on UNIX Systems

This table lists the shared libraries, include files, and other files installed on a UNIX system as part of a MATLAB C/C++ Graphics Library installation. In the table, `<matlab>` stands for your top-level MATLAB installation directory.

Table 1-4: List of Files Installed on UNIX Systems

File	Location	Description
libmwsgl.*ext*, where .*ext* is .so on Solaris systems and .sl on HP 700 systems	<matlab>/extern/lib/$ARCH where <matlab> is the name of your MATLAB root directory and $ARCH identifies the system architecture (i.e., alpha, lnx86, sgi, sgi64, sol2)	The graphics library binary file. The libraries are shared libraries for all platforms.
sgl sglcpp	<matlab>/bin	MATLAB Compiler bundle files, containing all the compiler command line options required to build a stand-alone graphics application.
libsgl.h libsglm.h	<matlab>/extern/include	The graphics library header file that contains prototypes for both the built-in and M-file graphics functions.
FigureMenuBar.fig FigureToolBar.fig	<matlab>/extern/include	Alternate menu bar and toolbar files used with the MATLAB figure window in stand-alone applications.
flames.m flames.mat	<matlab>/extern/examples/sgl	The M-file and MAT file for the graphics library example program.

Verifying Your Installation

To verify that the MATLAB C/C++ Graphics Library has been installed correctly, create a stand-alone application from the example M-file distributed with the library. Chapter 2, “Creating Stand-Alone Graphics Applications” describes how to build a stand-alone application. You can find the example program in the `<matlab>/extern/examples/sgl` directory.

Configuring the MATLAB C/C++ Graphics Library

After installing the MATLAB C/C++ Graphics Library, you should configure it using the `mbuild -setup` command. When you run `mbuild`, you specify:

- The ANSI C or C++ compiler you intend to use to compile the code generated by the MATLAB Compiler
- The libraries you want to link your application with; specifically, the MATLAB C/C++ Math Library alone, or the math library and the MATLAB C/C++ Graphics Library together.

This section includes information about:

- "Configuring the Graphics Library on PCs" on page 1-14
- "Configuring the Graphics Library on UNIX Systems" on page 1-16

For more information about the `mbuild` utility, see the *MATLAB Compiler User's Guide*. To learn how to build a stand-alone graphics application, see Chapter 2, "Creating Stand-Alone Graphics Applications".

Configuring the Graphics Library on PCs

To configure the graphics library on a PC running Microsoft Windows, run the `mbuild -setup` command. You can run `mbuild` at the MATLAB prompt or in a DOS Command Prompt window.

`mbuild` uses *options files* to specify all the compile and link command line options necessary to create a stand-alone graphics application using a particular compiler. When you configure the graphics library, you determine which options file `mbuild` uses to create stand-alone applications.

When you run `mbuild`, you specify the name and version of the compiler you intend to use. `mbuild` locates the options file specific to that compiler, and creates a copy of it in your system's user profiles directory. From then on, whenever the MATLAB Compiler calls `mbuild` to invoke your C or C++ compiler, it uses this local copy of the options file.

The following example illustrates running `mbuild -setup` on a PC. The example shows how to specify a compiler. If you have only one C or C++ compiler installed on your system, `mbuild` can determine its name and location automatically. To link with the graphics library, answer yes (`y`) at the `mbuild` prompt.

```
mbuild -setup

Please choose your compiler for building stand-alone MATLAB
applications.

Would you like mbuild to locate installed compilers [y]/n? n

Choose your C/C++ compiler:
[1] Borland C/C++          (version 5.0, 5.2, or 5.3)
[2] Microsoft Visual C/C++ (version 4.2, 5.2, or 6.0)
[3] Watcom C/C++           (version 10.6 or 11)

[0] None

Compiler: 2

Choose the version of your C/C++ compiler:
[1] Microsoft Visual C/C++ 4.2
[2] Microsoft Visual C/C++ 5.0
[3] Microsoft Visual C/C++ 6.0

version: 3

Your machine has a Microsoft Visual C/C++ compiler located at
D:\Program Files\DevStudio6.
Do you want to use this compiler [y]/n? y

Do you want to link against the C/C++ Graphics Library [y]/n? y

Please verify your choices:

Compiler: Microsoft Visual C/C++ 6.0
Location: D:\Program Files\DevStudio6
Linking against the C/C++ Graphics Library

Are these correct?([y]/n): y
```

Configuring the Graphics Library on UNIX Systems

To configure the graphics library on a UNIX system, run the `mbuild -setup` command. You can run `mbuild` at the MATLAB prompt or at the system prompt.

`mbuild` uses *options files* to specify all the compile and link command line options necessary to create a stand-alone graphics application. When you configure the graphics library, you specify the name of the options file you want to use.

On UNIX systems, `mbuild` presents a choice of two options files: `mbuildopts.sh` and `mbuildsglopts.sh`. To create a stand-alone graphics application, choose the `mbuildsglopts.sh` file (selection 2). When you select an options file, `mbuild` creates a local copy of the options file in your `~/matlab` directory.

Note Even though you select the `mbuildsglopts.sh` options file, when `mbuild` creates the local copy in `~/matlab`, it renames the file to `mbuildopts.sh`.

The following example illustrates running `mbuild` on a UNIX system. To link with the graphics library, select option 2. If you have run `mbuild` before, a local copy of the options file exists in your `~/matlab` directory. When `mbuild` asks if you want to overwrite this existing version of `mbuildopt.sh`, answer yes (`y`).

```
mbuild -setup

Using the 'mbuild -setup' command selects an options file that is
placed in ~/matlab and used by default for 'mbuild'. An options
file in the current working directory or specified on the command
line overrides the default options file in ~/matlab.

Options files control which compiler to use, the compiler and link
command options, and the run-time libraries to link against.

To override the default options file, use the 'mbuild -f' command
(see 'mbuild -help' for more information).

The options files available for mbuild are:

  1: /matlab/bin/mbuildopts.sh :
       Build and link with MATLAB C/C++ Math Library
  2: /matlab/bin/mbuildsglopts.sh :
      Build and link with MATLAB C/C++ Math and Graphics
            Libraries

Enter the number of the options file to use as your default options
file: 2
```

2

Creating Stand-Alone Graphics Applications

Introduction

After installing and configuring the MATLAB C/C++ Graphics Library, described in Chapter 1, “Introduction”, you can use it to create stand-alone graphics applications. This chapter includes the following topics:

Overview

To create a stand-alone C or C++ graphics application, use the MATLAB Compiler (`mcc`). The MATLAB Compiler:

- Translates the specified M-file into a C or C++ source code module
- Generates additional C or C++ source code modules, called *wrapper* files, required by stand-alone applications
- Compiles and links the source modules into a stand-alone application, by invoking your ANSI C or C++ compiler and linker.

Figure 2-1 shows how the Compiler works in conjunction with other tools and libraries to create a stand-alone graphics application. In the figure, each tool is represented by a shadowed box. For more detailed information about using the MATLAB Compiler and the `mbuild` utility, see the *MATLAB Compiler User's Guide*.

Note Do not confuse the MATLAB Compiler with an ANSI C or C++ compiler. The MATLAB Compiler translates M-code into C or C++ source code, which can then be compiled into object modules using an ANSI C or C++ compiler. This documentation distinguishes references to the MATLAB Compiler by using the word "Compiler" with a capital C. References to "compiler" with a lower case c refer to your ANSI C or C++ compiler.

Note that a stand-alone graphics application must be linked with several MATLAB libraries as well as an ANSI C/C++ math library. The MATLAB API and MAT-file libraries come with MATLAB. The MATLAB Math Built-In Library and the MATLAB M-File Math Library are components of the MATLAB C/C++ Math Library.

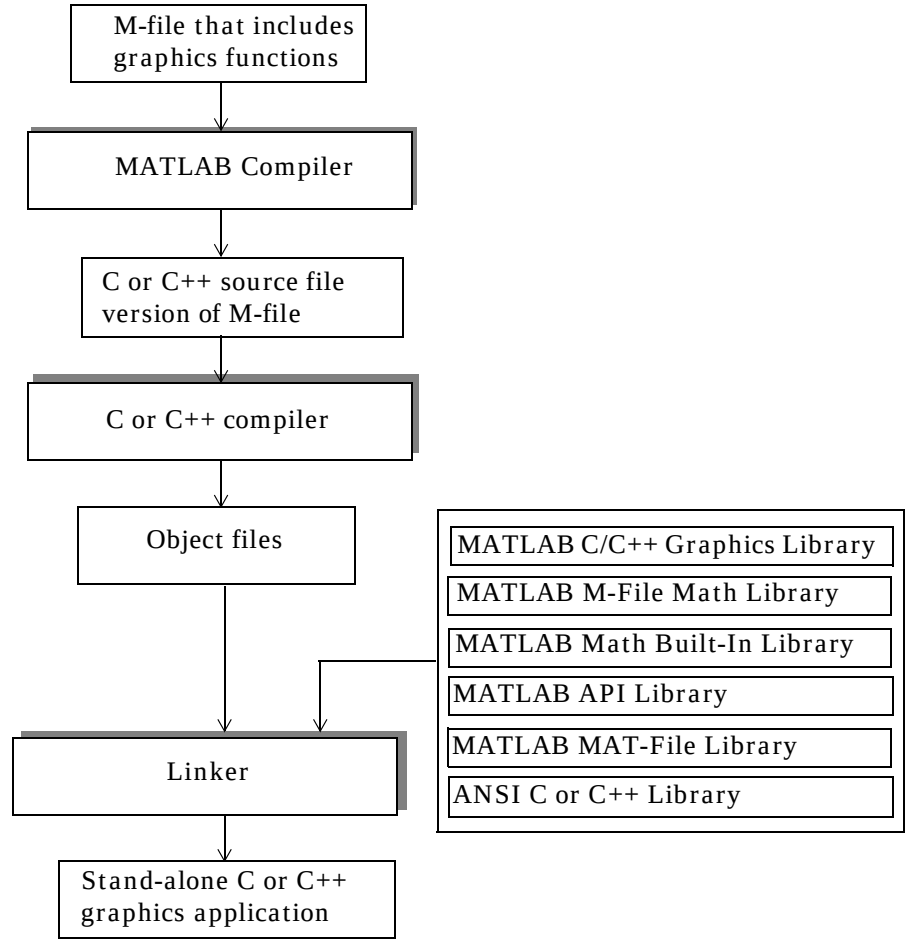

Figure 2-1: Creating a Stand-Alone C or C++ Graphics Applications

Building a Stand-Alone Graphics Application

The best way to learn how to build a stand-alone graphics application is to see an example. This section shows how to create a stand-alone graphics application by converting one of the demo programs included with MATLAB, `lorenz.m`. This demo plots the orbit of a point around the Lorenz chaotic attractor. (For more information about the Lorenz application, type `help lorenz` at the MATLAB prompt.) This demo is a useful example because it uses graphics functions and includes several user-interface objects, such as push buttons.

This section includes:

Building Graphics Applications on a PC

The following code example converts the Lorenz application into a stand-alone application on a PC. To try this example, start MATLAB and perform these steps at the MATLAB prompt. You can also create a stand-alone graphics application with running MATLAB. For information, see "Running the MATLAB Compiler Outside MATLAB" on page 2-9.

```
mbuild -setup

!copy <matlab>\toolbox\matlab\demos\lorenz.m *.*

mcc -B sgl lorenz.m

!lorenz
```

Note the following:

- The example uses `mbuild -setup` to set up the environment to create stand-alone applications. This step is only required the first time you create a stand-alone graphics application. For detailed information about running `mbuild`, see "Configuring the MATLAB C/C++ Graphics Library" on page 1-14.

- The example uses the DOS `copy` command to copy the Lorenz application M-file into the current MATLAB directory. `<matlab>` represents your toplevel MATLAB directory. (This step is suggested because you may not have permission to create a new file in the MATLAB `demos` directory.) You can also use Microsoft Windows Explorer to copy the file.
- The example invokes the MATLAB Compiler, using the `-B` flag to specify the graphics library *bundle* file. Bundle files are ASCII text files that contain Compiler command line options and arguments. The `sgl` bundle file, specified in the example, creates a C stand-alone application. To create a C++ application, use the `sglcpp` bundle file. The bundle files reside in the `<matlab>\bin` directory.

Note Do not use the Compiler `-V1.2` flag when creating a stand-alone graphics library.

Results of Compilation

When it translates an M-file application, the MATLAB Compiler generates multiple C or C++ source code modules in your current working directory. These include *wrapper* files that contain necessary components of a stand-alone application, such as a `main()` entry point.

In addition, the first time you run the MATLAB Compiler to create a stand-alone graphics application, it creates a subdirectory, named `\bin`, in your current working directory. The Compiler puts in this directory versions of the MATLAB menu bar and toolbar figure files that are used by stand-alone graphics applications at run-time. (For more information, see “Changes in Run-time Behavior and Appearance” on page 2-10.) When you run the Compiler subsequently, it checks for the existence of these files and does not overwrite them if they exist.

Running a Stand-Alone Graphics Application

The Compiler creates the stand-alone graphics application as an executable program in your current working directory, giving it the same name as your M-file but with the `.exe` filename extension. You can run the application at the MATLAB command prompt if you precede the name with a ! symbol, as shown in the example. You can also run stand-alone graphics applications outside the

MATLAB environment. However, make sure that the directory containing the shared libraries to which your application has been linked (<matlab>\bin) is on your directory search path.

Editing the Search Path on Windows 95. On Window95 systems, you must edit your autoexec.bat file to add your shared library directory to the PATH variable.

Editing the Search Path on Windows NT. On Windows NT systems, go the **Settings** option on the **Start** menu and choose **Control Panel**. Double-click on the System icon to view the **System Properties** dialog box. Use the Environment panel to edit the PATH variable.

Building Graphics Applications on a UNIX System

The following code example converts the Lorenz application into a stand-alone application on a UNIX system. To try this example, start MATLAB and perform these steps at the MATLAB prompt. You can also create a stand-alone graphics application with running MATLAB. For information, see "Running the MATLAB Compiler Outside MATLAB" on page 2-9.

```
mbuild -setup

!cp <matlab>/toolbox/matlab/demos/lorenz.m ./

mcc -B sgl lorenz.m

!lorenz
```

Note the following:

- The example uses mbuild -setup to set up the environment to create stand-alone applications. This step is only required the first time you create a stand-alone graphics application. For detailed information about running mbuild, see "Configuring the MATLAB C/C++ Graphics Library" on page 1-14.
- The example uses the UNIX cp command to copy the Lorenz application M-file into the current MATLAB directory. Use the ! symbol to execute an operating system command inside the MATLAB environment. (This step is suggested because you may not have permission to create a new file in the MATLAB demos directory.) <matlab> represents your toplevel MATLAB directory.

- The example invokes the MATLAB Compiler, using the `-B` flag to specify the graphics library *bundle* file. Bundle files are ASCII text files that contain Compiler command line options and arguments. The `sgl` bundle file, specified in the example, creates a C stand-alone application. To create a C++ application, use the `sglcpp` bundle file. The bundle files reside in the `<matlab>/bin` directory.

Note Do not use the `-V1.2` flag when using the MATLAB Compiler to create a stand-alone graphics library.

Results of Compilation

When it translates an M-file application, the MATLAB Compiler generates multiple C or C++ source code modules in your current working directory. These include *wrapper* files that contain necessary components of a stand-alone application, such as a `main()` entry point.

In addition, the first time you run the MATLAB Compiler to create a stand-alone graphics application, it creates a subdirectory, named `/bin`, in your current working directory. The Compiler puts in this directory versions of the MATLAB menu bar and toolbar figure files that are used by stand-alone graphics applications at run-time. (For more information, see “Changes in Run-time Behavior and Appearance” on page 2-10.) When you run the Compiler subsequently, it checks for the existence of these files and does not overwrite them if they exist.

Running a Stand-Alone Graphics Application

The Compiler creates the stand-alone graphics application as an executable program in your current working directory, giving it the same name as your M-file. You can run your stand-alone graphics application at the MATLAB prompt if you precede the executable name with a !, as shown in the example. You can also run a stand-alone application outside of the MATLAB environment. However, you must add to your path the location of the shared

libraries to which your application is linked. To set your path, use the command from this table that is specific for your system.

Architecture	Command
HP700	`setenv SHLIB_PATH <matlab>/extern/lib/hp700:<matlab>/bin/hp700:$SHLIB_PATH`
All others	`setenv LD_LIBRARY_PATH <matlab>/extern/lib/<arch>:<matlab>/bin/<arch>:$LD_LIBRARY_PATH`
	where: `<matlab>` is the MATLAB root directory. `<arch>` is your architecture (i.e., alpha, lnx86, sgi, sgi64, sol2).

To avoid having to reissue this command at the start of each login session, include it in a startup script such as `~/.cshrc` or `~/.login`. Use the `~/.login` option, if your system supports it, because it only gets executed once.

Running the MATLAB Compiler Outside MATLAB

You can run the MATLAB Compiler without also running MATLAB. However, if you do, you must specify the locations of the M-files that your application depends on, using the `-I` option on the Compiler command line. When you run the Compiler from within MATLAB, it can locate these files by referencing the MATLAB path.

For example, the Lorenz application uses functions in the `graph2d`, `graphics`, `demos`, and `graph3d` subdirectories of the `<matlab>/toolbox/matlab/` directory.

A convenient way to provide the Compiler with this path information is to start MATLAB and run the `mccsavepath` command. This command creates a file in your current working directory, named `mccpath`, that contains this path information. When you run the Compiler outside the MATLAB environment, it automatically looks for this path information file in your local directory.

Changes in Run-time Behavior and Appearance

Stand-alone versions of graphics applications typically look and operate the same as their M-file counterparts. Note, however, that because stand-alone applications run outside the MATLAB environment, some standard MATLAB menu bar options are not supported. For example, stand-alone applications cannot support plot editing or provide access to MATLAB Help files.

To illustrate these differences, compare Figure 2-2, which shows the Lorenz application running as an M-file on a PC, with Figure 2-3, which shows the Lorenz application running as a stand-alone application. These differences are discussed in the following sections:

- "Changes to Figure Window Menu Options" on page 2-11
- "Changes to Default Print Handling" on page 2-11
- "Including Help in a Stand-Alone Graphics Application" on page 2-12
- "Ctrl-C Handling" on page 2-12

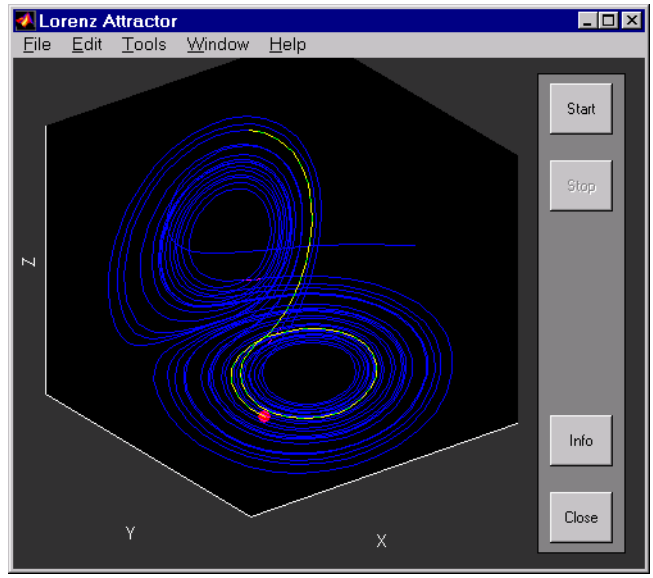

Figure 2-2: M-File Version of the Lorenz Application

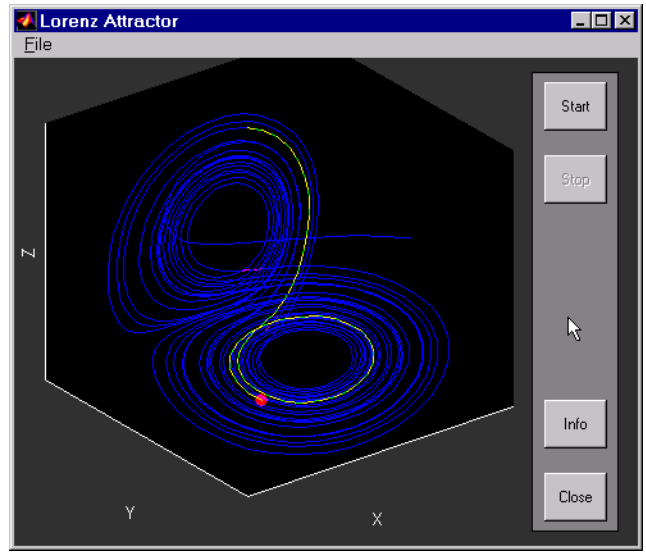

Figure 2-3: Stand-Alone Version of the Lorenz Application

Changes to Figure Window Menu Options

In stand-alone graphics applications, the Figure window menu bar contains only the **File** menu option. Stand-alone graphics applications run without MATLAB and cannot support many of the options available through the **Edit**, **Tools**, and **Help** menus, such as plot editing. When you create a stand-alone application, the graphics library excludes these items from the menu bar.

The graphics library also excludes options from the **File** menu, such as the **Property Editor** and Page Setup options, that are not supported by stand-alone applications.

Changes to Default Print Handling

Because stand-alone graphics applications support printing, the Figure window **File** menu includes the **Print** option. This option executes the `print` command setup when the application was created in MATLAB. For information about the various print options supported by the graphics library, see “Graphics Library Printing Support” on page 1-3.

Note, however, that the **Print** option in stand-alone graphics applications does not display the **Print** dialog box, as it does for M-file applications.

Including Help in a Stand-Alone Graphics Application

When you press the **Info** button in the M-file version of the Lorenz application, it displays a separate help window containing the M-file help for the Lorenz function. If you click on the **Info** button in the stand-alone version of the Lorenz application, you get the error message:

```
An error occurred in the callback : lorenz('info')
The error message caught was      : Function "helpwin" is not
     supported in standalone applications
```

Stand-alone applications do not have access to MATLAB help.

Ctrl-C Handling

When you run a graphics application within MATLAB, you can press **Ctrl-C** to break infinite loops. For example, you can press **Ctrl-C** to stop an animation. When you run a C or C++ stand-alone application, **Ctrl-C** handling is not supported.

Packaging a Graphics Application for Redistribution

To distribute a stand-alone graphics application, you must create a package that includes the application's executable as well as the shared libraries against which the application was linked. This section describes:

- "Packaging Graphics Applications on a PC" on page 2-13
- "Packaging Graphics Applications on a UNIX System" on page 2-14

Packaging Graphics Applications on a PC

When packaging an application for redistribution on a PC, you must include these libraries along with your application executable:

- `<matlab>\bin\sgl.dll`
- `<matlab>\bin\hg_sgl.dll`
- `<matlab>\bin\uiw_sgl.dll`
- `<matlab>\bin\gui_sgl.dll`
- `<matlab>\bin\hardcopy_sgl.dll`
- `<matlab>\bin\mpath.dll`
- `<matlab>\bin\libmmfile.dll`
- `<matlab>\bin\libmat.dll`
- `<matlab>\bin\libmcc.dll`
- `<matlab>\bin\libmatlb.dll`
- `<matlab>\bin\libmx.dll`
- `<matlab>\bin\libut.dll`

When packaging a stand-alone application, also remember to include:

- The ANSI C or C++ libraries you linked with your application
- The `\bin` directory associated with your stand-alone application. See "Results of Compilation" on page 2-6 for more information about this directory.
- All dynamic link libraries (DLLs) associated with your application must be on the system path. You must either install them in a directory that is already on the path or modify the `%PATH%` variable, as part of your application installation procedure, to include the new directory.

Packaging Graphics Applications on a UNIX System

When packaging an application for redistribution on a UNIX system, you must include these libraries with your application executable. In the filenames in this list, the *.ext* is `.so` on Solaris, Alpha, Linux, and SGI; and `.sl` on HP 700.

- `<matlab>/extern/lib/<arch>/libmwsgl.ext`
- `<matlab>/extern/lib/<arch>/libmat.ext`
- `<matlab>/extern/lib/<arch>/libmi.ext`
- `<matlab>/extern/lib/<arch>/libmmfile.ext`
- `<matlab>/extern/lib/<arch>/libmcc.ext`
- `<matlab>/extern/lib/<arch>/libmatlb.ext`
- `<matlab>/extern/lib/<arch>/libmx.ext`
- `<matlab>/extern/lib/<arch>/libut.ext`
- `<matlab>/bin/<arch>/libmwhg.ext`
- `<matlab>/bin/<arch>/libmwuix.ext`
- `<matlab>/bin/<arch>/libmwgui.ext`
- `<matlab>/bin/<arch>/libmwhardcopy.ext`
- `<matlab>/bin/<arch>/libmwmpath.ext`

When packaging a stand-alone application, also remember to include:

- The ANSI C or C++ library you linked with your application
- The contents of the `/bin` directory associated with your stand-alone application. See “Results of Compilation” on page 2-8 for more information about this directory.

Compiling and Linking Translated M-Files

It is possible to build a stand-alone application without using `mbuild`. You must still use the MATLAB Compiler to generate C or C++ source code modules. However, after that, you can compile and link these modules as you would any application.

Compiling Translated M-Files

Compile the generated C or C++ source files with an ANSI C or C++ compiler. You must make sure the include file search path includes the directory in which the file `matlab.h` resides. ANSI C or C++ compilers typically use the `-I` switch to add directories to the include file search path.

Linking with MATLAB Libraries

Link the resulting object files against these libraries:

- MATLAB C/C++ Graphics Library (`libmwsgl`)
- MATLAB M-File Math Library (`libmmfile`)
- MATLAB Math Built-In Library (`libmatlb`)
- MATLAB MAT-File Library (`libmat`)
- MATLAB API Library (`libmx`)
- ANSI C or C++ math library (`libm`)

On PCs. For a complete list of the libraries required for graphics applications, and the order in which they should be specified, start MATLAB and run the `mbuild -setup` command. When `mbuild` determines the C or C++ compiler you intend to use, it creates an options file for that compiler, called `compopts.bat`, that lists all the libraries, in the correct order. You can find the `compopts.bat` file in a subdirectory of the user profiles directory.

Note If you are using the Microsoft Visual C compiler, you must manually build import libraries from the `.def` files using the `lib` command. If you are using the Borland C compiler, you can link directly against the `.def` files using the `implib` command. If you are using the Watcom C compiler, you must build the import libraries from the DLLs using the `wlib` command. See your compiler documentation for information about these commands.

On UNIX Systems. To see a complete list of the libraries required for graphics applications, view the `mbuildsglopts.sh` file in `<matlab>/bin`. Search for the section of this file specific to your system architecture. The command lists the required libraries, specified in the correct order.

3

Troubleshooting

Introduction

The MATLAB Compiler can compile most M-files that use graphics into stand-alone applications. Some M-files, however, may include coding practices that are not supported by the Compiler or by the graphics library. In some cases, the MATLAB Compiler may not be able to translate the M-file into C or C++ code. In other cases, the M-file may compile successfully but fail when run as stand-alone application.

This chapter describes how to diagnose and correct these problems, including:

Using Unsupported MATLAB 5.3 Features

The MATLAB Compiler supports most of the MATLAB 5.3 language features, including multidimensional arrays, cell arrays, and structures. However, the Compiler does not support:

- The MATLAB `eval` or `input` command
- MATLAB objects
- MATLAB integer types (`int8`, `int16`, `int32`, `uint8`, `uint16`, and `uint32`)

See the *MATLAB Compiler User's Guide* for more information about these limitations.

Symptom

The Compiler outputs error messages that identify which unsupported feature prevented compilation.

Workaround

If your application uses unsupported features, the only workaround is to remove these unsupported features by recoding your application.

Compiling Applications Written as Scripts

The Compiler cannot compile applications written as scripts because scripts interact with the MATLAB base workspace and stand-alone applications do not have access to the MATLAB base workspace.

Symptom

If you attempt to compile a script, the Compiler outputs the error message

```
??? Error: File "filename" is a Script M-file and cannot be
compiled with the current Compiler.
```

where *filename* is the name of your script M-file.

Workaround

To compile an application written as a script, turn it into a MATLAB function. To do this, include the MATLAB function prototype at the top of the file. You must also find where the script depends on variables in the base workspace and declare these variables as global variables.

For example, in the following script, the variable `f`, set by the call to the `figure` function, exists in the base workspace. This variable is then passed as a parameter to the function, `my_func`, specified in the callback property string. Passing a workspace variable in a callback string is supported in MATLAB but it is not supported by the MATLAB Compiler.

```
f = figure;

p_btn = uicontrol(gcf,...
                  'style', 'pushbutton',...
                  'Position',[10 10 133 25 ],...
                  'String', 'Press Here',...
                  'CallBack','my_func(f);');
```

The following example shows this script transformed into a function.

```
function was_a_script()
% new function

global f;

f = figure;
```

```
p_btn = uicontrol(gcf,...
                  'style', ‘pushbutton',...
                  'Position',[10 10 133 25 ],...
                  'String', 'Press Here',...
                  'CallBack','my_callback’);
```

In this code example, note the following:

- The example changes the script into a function by including a MATLAB function prototype line at the top of the file.
- The example declares the variable `f`, formerly referenced in the base workspace, as a global variable. This makes it accessible to the callback routine.
- The example replaces the reference to `my_func` in the callback string with the name of a new function, `my_callback`. This new function performs the processing formerly done in the callback string.

  Here is the new callback function. Note how the function also declares `f` as a global variable.

  ```
  function my_callback()
  % revised callback

  global f;

  my_func(f);
  ```

Fixing Callback Problems: Illegal Syntax

When a user clicks on an interactive element in a user interface, such as a push button, MATLAB parses the text string associated with the `callback` property of the element and executes the commands contained in this string. The callback parser that comes with the MATLAB C/C++ Graphics Library supports a subset of M syntax supported by the MATLAB interpreter.

For example, both the MATLAB callback parser and the graphics library callback parser support these constructs in callback strings:

- Nesting function calls
- Passing scalar arguments

- Passing string arguments
- Passing non-nested array arguments
- Specifying ranges

This sample callback string illustrates a nested function call, `gca`; a string constant, `'color'`; a vector constant `[0 0 1]`; and a simple range, `0:.2:1`.

```
'set(gca, ''color'', [0 0 1], ''xtick'', 0:.2:1)'
```

These constructs, which are supported in MATLAB, are *not* supported by the graphics library callback parser:

- Passing multidimensional arrays as function arguments
- Referencing workspace variables
- Making assignments to workspace variables
- Performing math operations, such as addition or subtraction

Symptom

Your M-file application compiles successfully, creating a stand-alone executable. However, when you run it, interactive user interface elements, such as a push button, are not responsive. When you exit the application, the graphics library issues an error message such as:

```
An error occurred in the callback : colormap([ 1 1 1; 0 0 0]);
The error message caught was      : Colormap must have 3 columns:
                                      [R,G,B].
```

Workaround

To fix an application that includes any of these constructs in a callback string:

- Search the application M-file for the callback string associated with the callback property of the unresponsive user interface element. For information about finding callback strings, see “Finding Callback Strings in an M-File” on page 3-7.
- Remove the callback string from the M-file, if it uses unsupported programming constructs, and put it into a new M-file, using it as the basis for a new function.
- Specify the name of this new function as the value of the callback string associated with the user interface element.

For example, this graphics application displays the output of the peaks function and includes a push button that changes the colormap to black and white.

```
function my_test()
% Graphics library callback test application

peaks;

p_btn = uicontrol(gcf,...
                  'style', 'pushbutton',...
                  'Position',[10 10 133 25 ],...
                  'String', 'Make Black & White',...
                  'CallBack','colormap([1 1 1; 0 0 0]);');
```

This M-file compiles successfully, creating a stand-alone application. However, when you run it, the push button doesn't change the color map. When you close the application, the graphics library issues the error message:

```
An error occurred in the callback : colormap([ 1 1 1; 0 0 0]);
The error message caught was      : Colormap must have 3 columns:
                                       [R,G,B].
```

This error is generated because the array argument passed to the colormap function, [1 1 1; 0 0 0], contains more than one row. To fix this application, create a new function that contains the call to colormap:

```
function change_colormap()
% New function made from callback property string

colormap([1 1 1; 0 0 0]);
```

Then, in the application M-file, replace the callback string associated with the push button with the name of the new function, change_colormap:

```
function my_test()
% Graphics library callback test application

%#function change_colormap

peaks;

p_btn = uicontrol(gcf,...
```

```
'style', 'pushbutton',...
'Position',[10 10 133 25 ],...
'String', 'Make Black & White',...
'CallBack','change_colormap');
```

This modified M-file successfully compiles and, when executed as a stand-alone application, the push button operates as expected.

This example also includes a call to the `%#function` pragma. This pragma lists functions that you want the Compiler to compile. This is required if a function is specified only in a callback string and not anywhere else in an M-file. When processing an M-file, the Compiler determines which functions an M-file depends on and compiles them all. The Compiler does not check the callback strings for functions. For more information about this pragma, see "Fixing Callback Problems: Missing Functions" on page 3-7.

Finding Callback Strings in an M-File

To find callback strings in an M-file, search your M-file for the character strings `Callback` or `Fcn`. This will find all the `Callback` properties defined by Handle Graphics® objects, such as uicontrol and uimenu. In addition, this will find the properties of figures and axes that end in `Fcn`, such as `CloseRequestFcn`, that also support callbacks.

Fixing Callback Problems: Missing Functions

When the Compiler creates a stand-alone application, it compiles the M-file you specify on the command line and, in addition, it compiles any other M-files that your M-file calls. If your application includes a call to a function in a callback string or in a string passed as an argument to the `feval` function or an ODE solver, and this is the only place in your M-file this function is called, the Compiler will not compile the function. The Compiler does not look in these text strings for functions to compile.

Symptom

Your application runs but an interactive user interface element, such as a push button, is unresponsive. When you close the application, the graphics library issues this error message:

```
An error occurred in the callback : change_colormap
The error message caught was      : Reference to unknown function
                change_colormap from FEVAL in stand-alone mode.
```

Workaround

To eliminate this error, create a list all of the functions that are specified only in callback strings and pass this list to the `%#function` pragma. (See "Finding Missing Functions in an M-File" on page 3-8 for hints about finding functions in callback strings.) The Compiler processes any function listed in a `%#function` pragma.

For example, the call to the `change_colormap` function in the sample application, `my_test`, illustrates this problem. To make sure the Compiler processes the `change_colormap` M-file, list the function name in the `%#function` pragma.

```
function my_test()
% Graphics library callback test application

%#function change_colormap

peaks;

p_btn = uicontrol(gcf,...
                  'style', 'pushbutton',...
                  'Position',[10 10 133 25 ],...
                  'String', 'Make Black & White',...
                  'CallBack','change_colormap');
```

Note Instead of using the `%#function` pragma, you can specify the name of the missing M-file on the Compiler command line. For more information about this mechanisms, see the *MATLAB Compiler User's Guide*.

Finding Missing Functions in an M-File

To find functions that may need to be listed in a `%#function` pragma, check the text strings specified as callback strings or as arguments to `feval`, ODE solvers, `fmin`, `fmins`, `funm`, and `fzeros`.

To find text strings used as callback strings, search for the characters "Callback" or "fcn" in your M-file. This will find all the `Callback` properties defined by Handle Graphics® objects, such as uicontrol and uimenu. In

addition, this will find the properties of figures and axes that end in `Fcn`, such as `CloseRequestFcn`, that also support callbacks.

Depending on Graphics Settings in Start-Up Files

When you start MATLAB, it executes `startup.m`, if it exists. Your application may depend on Handle Graphics defaults that are set within a `startup.m` file.

Workaround

If your application depends on graphics settings in a `startup.m` file, include the `startup.m`, or the portion of it your application depends on, in the group of M-files that you compile with the MATLAB Compiler.

Print Option Does Not Appear on File Menu

If you create a stand-alone application and the **Print** option does not appear on the **File** menu, it may indicate that the menu bar and toolbar figure files in your application's `\bin` directory (`/bin` on UNIX systems) are not correct. The first time you create a graphics application, the Compiler creates this directory and populates it with two figure files, `FigureMenuBar.fig` and `FigureToolBar.fig`. After that, whenever you create graphics applications, the Compiler checks for the existence of these files and, if they exist, it does not replace them. Your application's `bin` directory may contain figure files from a previous release of the graphics library.

Workaround

Replace the menu bar and toolbar figure files in your application `bin` directory with the versions of these figure files in the MATLAB installation directory (`<matlab>\extern\include` on PCs or `<matlab>/extern/include` on UNIX systems). When you restart your stand-alone graphics application, it will use the new figure files.

Another way to replace your existing figure files with new figure files is to delete your application `bin` directory and run the Compiler. If this directory does not exist, the Compiler creates it and populates it with copies of the figure files stored in the MATLAB installation directory. There is no need to recompile your graphics M-file application, especially if this is a time-consuming task. Compiling a trivial M-file graphics application is enough to cause the creation of a new application `bin` directory.

4

Reference

MATLAB C/C++ Graphics Library

The MATLAB C/C++ Graphics Library is distributed as a single library composed of two groups of functions:

- MATLAB 5.3 built-in graphics functions
- MATLAB 5.3 M-file graphics functions

MATLAB 5.3 Built-In Graphics Functions

This table lists the built-in functions contained in the MATLAB C/C++ Graphics Library. The stand-alone versions behave as the functions do for MATLAB 5.3 except for the restrictions placed by the MATLAB Compiler 2.0.1 and MATLAB C and C++ Math Library 2.0.1.

Table 4-1: MATLAB 5.3 Built-In Functions in the Graphics Library

axes	contourc	copyobj
delete	dragrect	drawnow
figure	fill	fill3
findobj	frame2im	get
getframe	handle2struct	image
im2frame	ishandle	light
line	loglog	movie
patch	plot	plot3
rbbox	rectangle	reset
rmappdata	selectmoveresize	semilogx
semilogy	set	setappdata
struct2handle	surface	text
uicontrol	uigetfile	uimenu
uiputfile	uisetcolor	uisetfont
waitfor	waitforbuttonpress	

MATLAB 5.3 M-File Graphics Functions

This table lists the MATLAB M-file functions contained in the MATLAB C/C++ Graphics Library. The stand-alone versions of these functions behave as the functions do in MATLAB 5.3.

Table 4-2: MATLAB 5.3 M-File Graphics Functions in the Graphics Library

align	allchild	axis
brighten	camzoom	close
closereq	clruprop	colordef
colormap	colstyle	dialog
errordlg	filemenufcn	findall
gca	gcbf	gcbo
gcf	gco	getappdata
getuprop	ginput	hgload
hgsave	histc	hold
hsv	hsv2rgb	isappdata
ishold	jet	legend
makemenu	menubar	menulabel
moveaxis	msgbox	newplot
polar	print	questdlg
rotate3d	setptr	setuprop
textwrap	uiclearmode	uicontextmenu
uirestore	uiresume	uiwait
view	warndlg	watchoff
watchon	whitebg	zoom

Index

L

M

N

O

P

R

S

T

U

W

Z