MATLAB® C/C++ Graphics Library

The Language of Technical Computing

Computation

Visualization

Programming

The MATH WORKS Inc.

User's Guide

Version 2

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)
November 2000 First Printing Revised for Version 2.1 (Release 12)

Contents

Preface

1 Introduction

2 Creating Stand-Alone Graphics Applications

3 Troubleshooting

Preface

Using This Guide

Expected Background

This document is intended as a practical introduction to creating a stand-alone application that uses MATLAB's graphics features. It is written primariy for MATLAB programmers who want to create a stand-alone C or C++ application from a MATLAB application. Knowledge of common programming techniques is helpful, but is not essential.

To get the most out of this document, you should be familiar with the MATLAB Compiler. See the MATLAB Compiler documentation for more information.

Organization of the Document

Chapter	Description
Chapter 1	Provides an overview of the graphics library and its components. This chapter also describes how to configure the graphics library.
Chapter 2	Describes how to use the graphics library to build a stand-alone application. This chapter includes information about packaging your application for redistribution.
Chapter 3	Provides important troubleshooting tips.

Typographical Conventions

We use some or all of these conventions in our manuals.

Item	Convention to Use	Example
Example code	Monospace font	To assign the value 5 to A, enter `A = 5`
Function names/syntax	Monospace font	The cos function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Keys	**Boldface** with an initial capital letter	Press the **Return** key.
Literal strings (in syntax descriptions in Reference chapters)	**`Monospace bold`** for literals.	`f = freqspace(n,'whole')`
Mathematical expressions	Variables in *italics* Functions, operators, and constants in standard text.	This vector represents the polynomial $p = x^2 + 2x + 3$

MATLAB output	Monospace font	MATLAB responds with
		`A =`
		`  5`

Menu names, menu items, and controls	**Boldface** with an initial capital letter	Choose the **File** menu.
New terms	*Italics*	An *array* is an ordered collection of information.
String variables (from a finite list)	*Monospace italics*	`sysc = d2c(sysd, 'method')`

Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the MATLAB C/C++ Graphics Library.

For more information about any of these products, see either:

- The online documentation for that product, if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at `http://www.mathworks.com`; see the "products" section.

Product	Description
MATLAB	Integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language
MATLAB Compiler	Compiler for automatically converting MATLAB M-files to C and C++ code
MATLAB C/C++ Math Library	Library that makes MATLAB's math functions available to developers of C and C++ applications

1

Introduction

The MATLAB® C/C++ Graphics Library is a collection of MATLAB graphics routines distributed as a single library. The graphics library makes the MATLAB plotting and visualization capabilities 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 custom MATLAB applications.

Using the graphics library with the MATLAB Compiler and the MATLAB C/C++ Math Library, you can compile M-files that include lines, text, meshes, and polygons, as well as graphical user interface (GUI) components such as menus, push buttons, and dialog boxes.

Note You may freely distribute applications you develop with the MATLAB C/C+ Graphics Library, subject to The MathWorks software license agreement.

This chapter includes the following topics:

After reading these sections, see "Configuring the MATLAB C/C++ Graphics Library" on page 1-10.

Components of the MATLAB C/C++ Graphics Library

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

- MATLAB 6.0 built-in graphics functions, such as `surf`, `plot`, `get`, and `set`
- Some commonly used MATLAB 6.0 M-file graphics functions, such as `newplot`, `gcf`, `gca`, `gco`, and `gcbf`

Note The MATLAB C/C+ Graphics Library includes only a subset of MATLAB M-file graphics functions. If you application includes graphics functions that are not included in the library, the MATLAB Compiler will compile them when it creates your application. For information about MATLAB features that are not supported by the Graphics Library, see "Restrictions" on page 1-6.

Files Installed on PCs

Table 1-1 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.

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

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

Table 1-1: 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>\toolbox\compiler\bundles	MATLAB Compiler bundle files, containing all the compiler options required to build a stand-alone graphics application.
libsgl.h libmwsglm.h libmwsglm.mlib 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.

Files Installed on UNIX Systems

Table 1-2 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.

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

Table 1-2: 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>` `<ARCH>` identifies the system architecture (i.e., `alpha`, `glnx86`, `sgi`, `sgi64`, `sol2`)	The graphics library binary file. The libraries are shared libraries for all platforms.
`sgl` `sglcpp`	`<MATLAB>/bin/toolbox/compiler/bundles`	MATLAB Compiler bundle files, containing all the compiler command line options required to build a stand-alone graphics application.

`libsgl.h`
`libmwsglm.h`
`libmwsglm.mlib`
	`<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.

Restrictions

The MATLAB C/C++ Graphics Library supports most MATLAB 6.0 features, including multidimensional arrays, cell arrays, and structures. However, there are some MATLAB features the graphics library does not support, including:

- MATLAB objects
- MATLAB Java objects
- `plotedit` command
- `propedit` command

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

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

Graphics Library Printing Support

If your application uses default `print` command settings, it should require no modification. The default `print` command sends the current figure to whatever printer has been set up as the default printer on the system on which your application is running. However, 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, but it does not support any of the Ghostscript drivers.

Table 1-3 lists the device drivers supported by the graphics library. For more information about specifying device drivers, see the “Printing MATLAB Graphics” section in MATLAB Graphics documentation.

Table 1-3: 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 other command-line options that control various aspects of the print job, such as the renderer used. Table 1-4 lists the subset of these options supported by

the graphics library. For a complete list of print command options, see the "Printing MATLAB Graphics" section in MATLAB graphics documentation.

Table 1-4: 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 (dpi).
`-painters`	Render using Painter's algorithm.
`-zbuffer`	Render using Z-buffer.

Unsupported Application Coding Practices

The graphics library does not support certain graphics M-file coding practices that are supported in the MATLAB interpreted environment. For example, MATLAB allows you to specify variable references and assignments in callback property strings. The graphics library does not support this coding practice in stand-alone graphics applications.

See Chapter 3, "Troubleshooting" for more information about unsupported coding practices and how to work around them.

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 6.0
- MATLAB Compiler Version 2.1
- MATLAB C/C++ Math Library Version 2.1

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

Hardware. The MATLAB C/C++ Graphics Library is available for these platforms:

- PCs running Microsoft Windows or Linux
- Sun
- HP
- SGI
- Compaq Alpha UNIX platforms

On HP systems, you need an ANSI C++ compiler (`aCC`) to create stand-alone graphics applications, even if you are working in C.

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 12, see the System Requirements page in the Products area of the MathWorks Web site, `www.mathworks.com`.

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 the following topics:

- Configuring the Graphics Library on PCs
- "Configuring the Graphics Library on UNIX Systems" on page 1-12

After configuring the graphics Library, see Chapter 2, "Creating Stand-Alone Graphics Applications", to learn how to use it to build a stand-alone graphics application.

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.

This example illustrates how to specify a compiler running `mbuild -setup` on a PC. `mbuild` can also determine the name and location of your C or C++ compiler automatically. To link with the graphics library, answer yes (y) when `mbuild` prompts you.

```
Please choose your compiler for building standalone MATLAB
applications:

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

Select a compiler:
[1] Borland C++Builder version 5.0
[2] Borland C++Builder version 4.0
[3] Borland C++Builder version 3.0
[4] Borland C/C++ version 5.02
[5] Borland C/C++ version 5.0
[6] Borland C/C++ (free command line tools) version 5.5
[7] Lcc C version 2.4
[8] Microsoft Visual C/C++ version 6.0
[9] Microsoft Visual C/C++ version 5.0
[10] Microsoft Visual C/C++ version 4.2

[0] None

Compiler: 9

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

Would you like 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
```

When you press **Enter**, `mbuild` creates an options file in your `C:\WINNT\Profiles` directory.

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

This chapter describes how to use the MATLAB C/C++ Graphics Library to build stand-alone graphics applications. It provides:

- An overview of the process
- Detailed instructions on how to build a stand-alone graphics application on PCs or UNIX systems, including an example
- A summary of the differences in the run-time appearance and behavior of M-file applications and their stand-alone counterparts
- Information about packaging a stand-alone graphics application for redistribution

This chapter includes information about the MATLAB Compiler because it is a necessary part of the process. For more detailed information about the Compiler, see the MATLAB Compiler documentation.

Overview

You use the MATLAB Compiler (mcc) to create a stand-alone C or C++ graphics application. In this process, the MATLAB Compiler:

- Translates the specified M-files into a C or C++ source code modules
- 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 an ANSI C or C++ compiler and linker that you have installed on your system.

  The Compiler links your application with the MATLAB C/C++ Graphics Library, several other MATLAB libraries, and 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 Math M-file Library are components of the MATLAB C/C++ Math Library. Figure 2-1 graphically illustrates this process.

Note To avoid confusion between the MATLAB Compiler and an ANSI C or C++ compiler, this documentation uses “Compiler” with a capital C to refer to the MATLAB Compiler and “compiler” with a lowercase c refer to an ANSI C or C++ compiler.

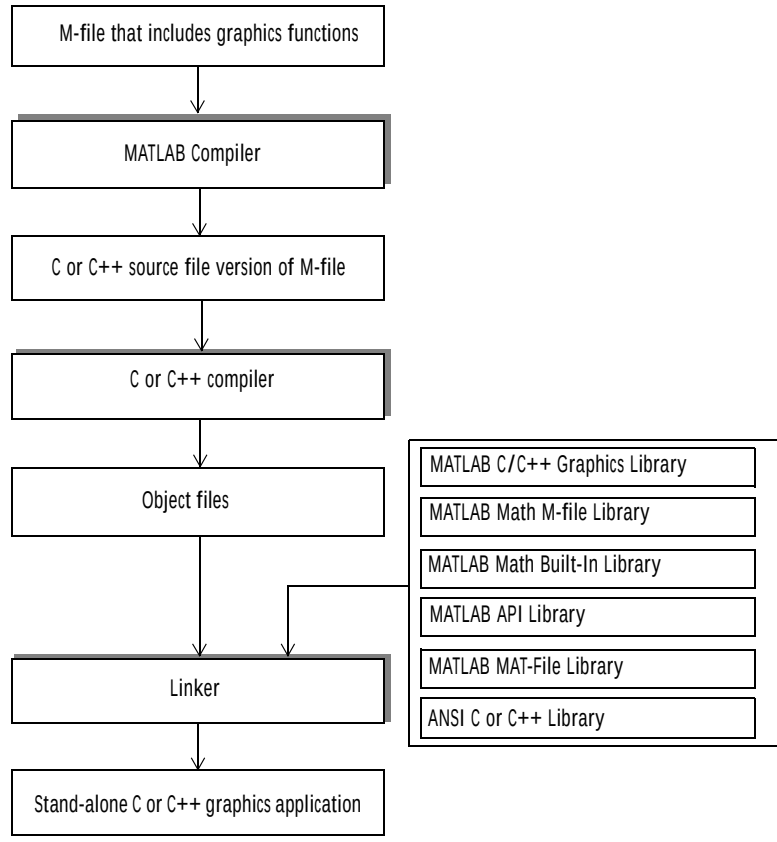

Figure 2-1: Creating a Stand-Alone C or C++ Graphics Applications

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`. The Lorenz demo program is a good example of a graphics application because it uses graphics functions and includes several user-interface objects, such as push buttons. (To get more information about the Lorenz application, type `help lorenz` at the MATLAB prompt.)

This section includes these topics:

Building Graphics Applications on a PC

To create a stand-alone graphics application on a PC, you must use the MATLAB Compiler (`mcc`), specifying the Compiler's Graphics Library *bundle* file.

Bundle files are ASCII text files that contain Compiler command line options and arguments. The MATLAB Compiler supports separate bundle files for creating C stand-alone graphics applications and C++ applications.

MATLAB Compiler's Graphics Library Bundle Files on PCs

C applications	`<MATLAB>\toolbox\compiler\bundles\sgl`
C++ applications	`<MATLAB>\toolbox\compiler\bundles\sglcpp`

For example, to convert the Lorenz application into a stand-alone application, enter these commands at the MATLAB prompt.

```
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 is only required the first time you create a stand-alone graphics application. See "Configuring the MATLAB C/C++ Graphics Library" on page 1-10 for more information about mbuild.
- The example uses the DOS `copy` command to copy the Lorenz application M-file into the current MATLAB directory. Replace `<MATLAB>` with the name of your top-level MATLAB installation directory. (This 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 bundle file used to create C stand-alone graphics applications, `sgl`.

Results of Compilation

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. (Stand-alone graphics applications use a special menu bar and toolbar. For more information, see "Changes in Run-Time Behavior and Appearance" on page 2-11.) Subsequently, whenever you run the Compiler, it checks for the existence of these files in the `\bin` directory 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 Window 95 systems, you must edit your `autoexec.bat` file to add the 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

To create a stand-alone graphics application on a UNIX system, you must use the MATLAB Compiler (`mcc`), specifying the Compiler's Graphics Library *bundle* file.

Bundle files are ASCII text files that contain Compiler command line options and arguments. The MATLAB Compiler supports separate bundle files for creating C stand-alone graphics applications and C++ applications.

MATLAB Compiler's Graphics Library Bundle Files on UNIX Systems

C applications	`<MATLAB>/toolbox/compiler/bundles/sgl`
C++ applications	`<MATLAB>/toolbox/compiler/bundles/sglcpp`

For example, to convert the Lorenz application into a stand-alone application, enter these commands at the MATLAB prompt.

```
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 is only required the first time you create a stand-alone graphics application. See "Configuring the MATLAB C/C++ Graphics Library" on page 1-10 for more information about `mbuild`.

- 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 is suggested because you may not have permission to create a new file in the MATLAB `demos` directory.) Replace `<MATLAB>` with the name of your top-level MATLAB installation directory.
- The example invokes the MATLAB Compiler, using the `-B` flag to specify the bundle used to create C stand-alone graphics applications, `sgl`

Results of Compilation

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. (Stand-alone graphics applications use a special menu bar and toolbar. For more information, see "Changes in Run-Time Behavior and Appearance" on page 2-11.) Subsequently, when you run the Compiler, it checks for the existence of these files in the `/bin` directory 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 system architecture

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 outside the MATLAB environment, invoking it at the system prompt. If you do, you must use the `-I` option on the Compiler command line to specify the locations of the M-files that your application depends on. For example, the Lorenz application uses functions in the `graph2d`, `graphics`, `demos`, and `graph3d` subdirectories of the `<MATLAB>/toolbox/matlab/` directory. When you run the Compiler from within MATLAB, it can locate these files by referencing the MATLAB path.

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

Compiling and Linking Without mbuild

For graphics applications, you must use the MATLAB Compiler to generate C or C++ source code modules. The graphics library does not support the direct coding of graphics applications. You can, however, perform the compilation and linking of your source modules without using `mbuild`.

To determine the libraries you need to link with, use the `mbuild` command with the `-n` option. When you specify this option, `mbuild` sets up the compile and link command lines necessary to build a stand-alone application but does not execute the commands. View the output of `mbuild -n` to determine the list of libraries you must link your application with and the order in which you must specify them.

You can also specify this `mbuild` option on the MATLAB Compiler command line by specifying `-M` option.

```
mcc -M -n -B sgl lorenz.m
```

Note On PCs, 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. See your compiler documentation for information about these commands.

Changes in Run-Time Behavior and Appearance

Stand-alone versions of graphics applications typically look and operate the same as their M-file counterparts. However, because stand-alone applications run outside the MATLAB environment, there are some differences, highlighted in these sections:

Changes to Figure Window Menu Bar Options

Stand-alone graphics applications use a special version of the Figure window menu bar that contains only the **File** menu option. The graphics library excludes the other standard menu bar items, such as **Edit**, **Tools**, and **Help**, from the menu bar because stand-alone graphics applications cannot support many of the options available through these menus.

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.

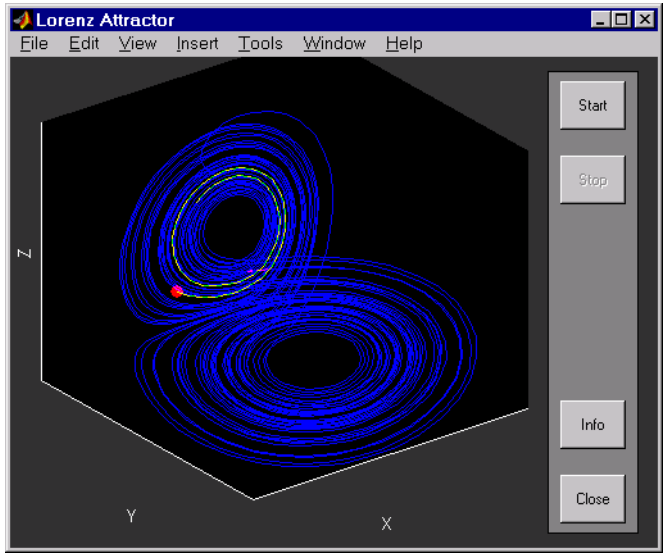

Figure 2-2: M-File Version of the Lorenz Application

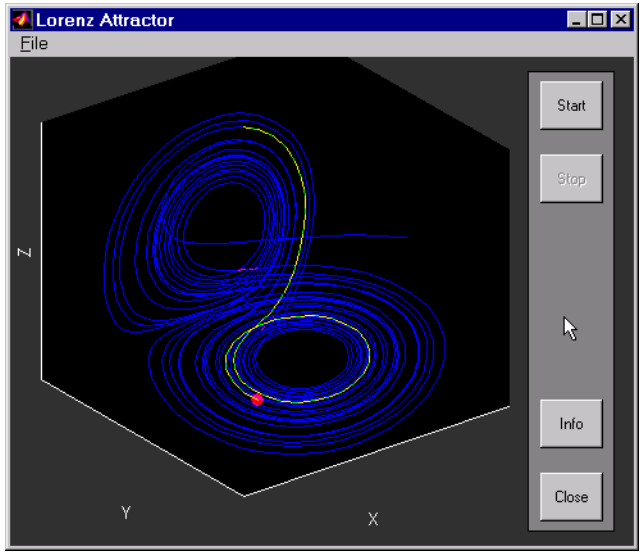

Figure 2-3: Stand-Alone Version of the Lorenz Application

Changes to the Figure Window File Menu Options

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

Note The graphics library includes the **Print** option in the **File** menu of stand-alone graphics applications. However, the **Print** option in stand-alone applications does not display the **Print** dialog box, as it does for M-file applications.

Accessing Help in Stand-Alone Applications

Some M-file applications include GUI components that provide access to MATLAB help. For example, the Lorenz application includes an **Info** push button that displays the M-file help for the Lorenz function in a separate window.

The stand-alone version of the Lorenz application does not have access to MATLAB help files. If you click on the **Info** button, you get this error message:

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

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.

Distributing Stand-Alone Graphics Applications

You may freely distribute applications you develop with the MATLAB C/C++ Graphics Library, subject to The MathWorks software license agreement. However, when you package your application for distribution, remember to include, along with your application executable, these additional files:

- The contents, if any, of a directory named `bin`, created by `mbuild` in the same directory as your application executable
- Any custom MEX files your application uses
- All the MATLAB math and graphics run-time libraries

To make packaging an application easier, the graphics library has prepackaged all the necessary MATLAB run-time libraries into a single, self-extracting archive file. For more information about how you can use this archive, see "Packaging the MATLAB Run-Time Libraries". For information about how customers who receive your application can use this archive, see "Installing Your Application" on page 2-15.

Packaging the MATLAB Run-Time Libraries

The MATLAB C/C++ Graphics Library has prepackaged all the MATLAB math and graphics run-time libraries required by stand-alone graphics applications into a single, self-extracting archive file, called the MATLAB Math and Graphics Run-Time Library Installer. Instead of including all the run-time libraries individually in your stand-alone application distribution package, you can simply include this archive file.

The following table lists the name of the archive file for both PCs and UNIX systems. In the table `<MATLAB>` represents your MATLAB installation directory and `<ARCH>` represents your UNIX platform.

Platform	MATLAB Math and Graphics Run-Time Library Installer
UNIX systems	`<MATLAB>/extern/lib/<ARCH>/mglinstaller`
PCs	`<MATLAB>\extern\lib\win32\mglinstaller.exe`

Installing Your Application

To install your application, your customers must:

- Run the MATLAB Math and Graphics Run-Time Library Installer. This program extracts the libraries from the archive and installs them in subdirectories of a directory specified by the user.
- Add the `bin/$ARCH` subdirectory to their path. This is the only MATLAB run-time library subdirectory that needs to be added to the path.

Note If a customer already has the MATLAB math and graphics run-time libraries installed on their system, they do not need to reinstall them. They only need to ensure that the library search path is configured correctly.

On UNIX Systems

On UNIX systems, your customers run the MATLAB Math and Graphics Run-Time Library Installer by executing the `mglinstaller` command at the system prompt. Your customers can specify the name of the directory into which they want to install the libraries. By default, the installer puts the files in the current directory.

After the installer unpacks and uncompresses the libraries, your customers must add the name of the `bin/<ARCH>` subdirectory to the `LD_LIBRARY_PATH` environment variable. (The equivalent variable on HP-UX systems is the `SHLIB_PATH`.)

For example, if your customers working on a Linux system specify the installation directory `mgl_runtime_dir`, then they must add `mgl_runtime_dir/bin/glnx86` to the `LD_LIBRARY_PATH` environment variable.

On PCs

On PCs, your customers can run the MATLAB Math and Graphics Run-Time Library Installer by double-clicking on the `mglinstaller.exe` file. Your customers can specify the name of the directory into which they want to install the libraries. By default, the installer puts the files in the current directory.

After the installer unpacks and uncompresses the libraries, your customers must add the `bin\win32` subdirectory to the system path variable (`PATH`).

For example, if your customers specify the installation directory `mgl_runtime_dir`, then they must add `mgl_runtime_dir\bin\win32` to `PATH`.

3

Troubleshooting

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 a stand-alone application.

This chapter describes how to diagnose and correct these problems. It includes these sections:

Using Unsupported MATLAB 6.0 Features

The MATLAB Compiler supports most of the MATLAB 6.0 language features, including multidimensional arrays, cell arrays, and structures. However, the Compiler does not support:

- Certain uses of the MATLAB `eval` or `input` command.
- MATLAB objects
- MATLAB Java objects

See the MATLAB Compiler documentation 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 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: 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 the names of 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 3-7 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 documentation.

Finding Missing Functions in an M-File

To find functions in your application that may need to be listed in a `%#function` pragma, search your M-file source code for text strings specified as callback strings or as arguments to the `feval`, `fminbnd`, `fminsearch`, `funm`, and `fzero` functions or any ODE solvers.

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.

File Menu Does Not Appear in Application

If you create a stand-alone application and the **File** menu does not appear in the menu bar, 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.

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. Your application must explicitly call these graphics settings.

Problem Starting Graphics Application Executable

If you are using Microsoft Visual C++ 5.0, or have the Microsoft Foundation Class (MFC) DLL from MSVC 5.0 installed in your system directory, you may encounter a problem starting a graphics library application.

Note Customers of your application may encounter the same problem if they have this DLL installed on their system.

Symptom

Your application compiles and an executable is created. However, when you invoke it, no figure window appears. If you run the application from a DOS command window, you or one of your customers may see an error message such as:

```
The UIW_SGL.DLL file is linked to missing export MFC42.DLL:####.
```

or

```
The ordinal #### could not be located in the dynamic-link library
MFC42.dll.
```

Workaround

To fix this problem, locate the files named `MFC42.dll` and `MSVCRT.dll` in your Windows system directory and replace them with the versions of these files in the `<MATLAB>\bin\win32` directory, where `<MATLAB>` represents the name of your MATLAB installation directory.

Your customers who encounter the same problem should also replace the versions of these files in the Windows system directory. They can use the versions they find in `<MGLRUNTIMELIBRARY>\bin\win32` directory, where `<MGLRUNTIMELIBRARY>` is the name of the directory in which they installed the MATLAB Math and Graphics Run-Time Libraries. See "Distributing Stand-Alone Graphics Applications" on page 2-14 for more information.

You may also encounter this problem with the files `dforrt.dll` or `dformd.dll`. You can replace these files in the Windows system directory with their counterparts in the `<MGLRUNTIMELIBRARY>\bin\win32` directory.

Index

G

H

I

J

L

M

O

P

R

S

T

U

W

Z