Software introduction:

**1. Introduction to MATLAB:**

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

* Math and computation
* Algorithm development
* Data acquisition
* Modeling, simulation, and prototyping
* Data analysis, exploration, and visualization
* Scientific and engineering graphics
* Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most uses of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M – files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

**1.1 The MATLAB system:**

The MATLAB system consists of five main parts

* **Development Environment**:

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and command window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

* **The MATLAB Mathematical Function Library**:

This is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigen values, Bessel functions, and fast Fourier transforms.

* **The MATLAB Language**:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both “programming in the small” to rapidly create quick and dirty throw-away programs, and “programming in the large” to create large and complex application programs.

* **Graphics:**

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

* **The MATLAB Application Program Interface (API):**

This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Various toolboxes are there in MATLAB for computing recognition techniques, but we are using **IMAGE PROCESSING** toolbox.

**1.2 GRAPHICAL USER INTERFACE (GUI):**

MATLAB’s Graphical User Interface Development Environment (GUIDE) provides a rich set of tools for incorporating graphical user interfaces (GUIs) in M-functions. Using GUIDE, the processes of laying out a GUI (i.e., its buttons, pop-up menus, etc.)and programming the operation of the GUI are divided conveniently into two easily managed and relatively independent tasks. The resulting graphical M-function is composed of two identically named (ignoring extensions) files:

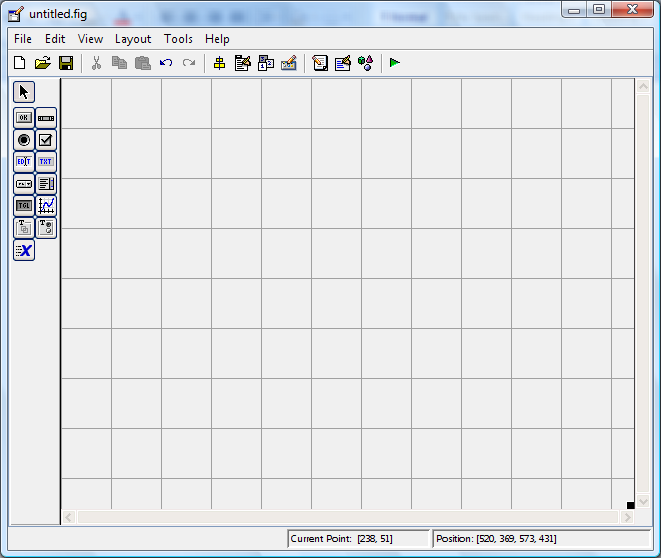
* A file with extension .fig, called a FIG-file that contains a complete graphical description of all the function’s GUI objects or elements and their spatial arrangement. A FIG-file contains binary data that does not need to be parsed when he associated GUI-based M-function is executed.
* A file with extension .m, called a GUI M-file, which contains the code that controls the GUI operation. This file includes functions that are called when the GUI is launched and exited, and callback functions that are executed when a user interacts with GUI objects for example, when a button is pushed.

To launch GUIDE from the MATLAB command window, type

guide filename

Where filename is the name of an existing FIG-file on the current path. If filename is omitted,

GUIDE opens a new (i.e., blank) window.



A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called components that enable a user to perform interactive tasks. The user of the GUI does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user of a GUI need not understand the details of how the tasks are performed.

GUI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders just to name a few. GUIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other GUIs, and display data as tables or as plots.

**software description**

**Getting Started**

If you are new to MATLAB, you should start by reading Manipulating Matrices. The most important things to learn are how to enter matrices, how to use the: (colon) operator, and how to invoke functions. After you master the basics, you should read the rest of the sections below and run the demos.

At the heart of MATLAB is a new language you must learn before you can fully exploit its power. You can learn the basics of MATLAB quickly, and mastery comes shortly after. You will be rewarded with high productivity, high-creativity computing power that will change the way you work.

**Introduction** - describes the components of the MATLAB system.

**Development Environment** - introduces the MATLAB development environment, including information about tools and the MATLAB desktop.

**Manipulating Matrices** - introduces how to use MATLAB to generate matrices and perform mathematical operations on matrices.

**Graphics** - introduces MATLAB graphic capabilities, including information about plotting data, annotating graphs, and working with images.

**Programming with MATLAB** - describes how to use the MATLAB language to create scripts and functions, and manipulate data structures, such as cell arrays and multidimensional arrays.

**INTRODUCTION**

What Is MATLAB?

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

* Math and computation
* Algorithm development
* Modeling, simulation, and prototyping
* Data analysis, exploration, and visualization
* Scientific and engineering graphics
* Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

**The MATLAB System**

The MATLAB system consists of five main parts:

**Development Environment**. This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, and browsers for viewing help, the workspace, files, and the search path.

**The MATLAB Mathematical Function Library**. This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

**The MATLAB Language**. This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

**Handle Graphics®**. This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

**The MATLAB Application Program Interface (API).** This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

**DEVELOPMENT ENVIRONMENT**

**Introduction**

This chapter provides a brief introduction to starting and quitting MATLAB, and the tools and functions that help you to work with MATLAB variables and files. For more information about the topics covered here, see the corresponding topics under Development Environment in the MATLAB documentation, which is available online as well as in print.

**Starting and Quitting MATLAB**

**Starting MATLAB**

On a Microsoft Windows platform, to start MATLAB, double-click the MATLAB shortcut icon on your Windows desktop.

On a UNIX platform, to start MATLAB, type matlab at the operating system prompt.

After starting MATLAB, the MATLAB desktop opens - see MATLAB Desktop.

You can change the directory in which MATLAB starts, define startup options including running a script upon startup, and reduce startup time in some situations.

**Quitting MATLAB**

To end your MATLAB session, select Exit MATLAB from the File menu in the desktop, or type quit in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a finish.m script.

**MATLAB Desktop**

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries.

You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

You can specify certain characteristics for the desktop tools by selecting Preferences from the File menu. For example, you can specify the font characteristics for Command Window text. For more information, click the Help button in the Preferences dialog box.

**Desktop Tools**

This section provides an introduction to MATLAB's desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are:

* Current Directory Browser
* Workspace Browser
* Array Editor
* Editor/Debugger
* Command Window
* Command History
* Launch Pad
* Help Browser

**Command Window**

Use the Command Window to enter variables and run functions and M-files.

**Command History**

Lines you enter in the Command Window are logged in the Command History window. In the Command History, you can view previously used functions, and copy and execute selected lines. To save the input and output from a MATLAB session to a file, use the diary function.

**Running External Programs**

You can run external programs from the MATLAB Command Window. The exclamation point character! is a shell escape and indicates that the rest of the input line is a command to the operating system. This is useful for invoking utilities or running other programs without quitting MATLAB. On Linux, for example,!emacs magik.m invokes an editor called emacs for a file named magik.m. When you quit the external program, the operating system returns control to MATLAB.

**Launch Pad**

MATLAB's Launch Pad provides easy access to tools, demos, and documentation.

**Help Browser**

Use the Help browser to search and view documentation for all your Math Works products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.

To open the Help browser, click the help button in the toolbar, or type helpbrowser in the Command Window. The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information.

**Help Navigator**

Use to Help Navigator to find information. It includes:

**Product filter** - Set the filter to show documentation only for the products you specify.

**Contents tab** - View the titles and tables of contents of documentation for your products.

**Index tab** - Find specific index entries (selected keywords) in the MathWorks documentation for your products.

**Search tab** - Look for a specific phrase in the documentation. To get help for a specific function, set the Search type to Function Name.

**Favorites tab** - View a list of documents you previously designated as favorites.

**Display Pane**

After finding documentation using the Help Navigator, view it in the display pane. While viewing the documentation, you can:

**Browse to other pages** - Use the arrows at the tops and bottoms of the pages, or use the back and forward buttons in the toolbar.

**Bookmark pages** - Click the Add to Favorites button in the toolbar.

**Print pages** - Click the print button in the toolbar.

**Find a term in the page** - Type a term in the Find in page field in the toolbar and click Go.

Other features available in the display pane are: copying information, evaluating a selection, and viewing Web pages.

**Current Directory Browser**

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.

**Search Path**

To determine how to execute functions you call, MATLAB uses a search path to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and MathWorks toolboxes are included in the search path.

**Workspace Browser**

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.

To view the workspace and information about each variable, use the Workspace browser, or use the functions who and whos.

To delete variables from the workspace, select the variable and select Delete from the Edit menu. Alternatively, use the clear function.

The workspace is not maintained after you end the MATLAB session. To save the workspace to a file that can be read during a later MATLAB session, select Save Workspace As from the File menu, or use the save function. This saves the workspace to a binary file called a MAT-file, which has a .mat extension. There are options for saving to different formats. To read in a MAT-file, select Import Data from the File menu, or use the load function.

**Array Editor**

Double-click on a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.

**Editor/Debugger**

Use the Editor/Debugger to create and debug M-files, which are programs you write to runMATLAB functions. The Editor/Debugger provides a graphical user interface for basic textediting, as well as for M-file debugging.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.

**MANIPULATING MATRICES**

**Entering Matrices**

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways:

* Enter an explicit list of elements.
* Load matrices from external data files.
* Generate matrices using built-in functions.
* Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You have only to follow a few basic conventions:

* Separate the elements of a row with blanks or commas.
* Use a semicolon, ; , to indicate the end of each row.
* Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]

MATLAB displays the matrix you just entered.

A =

16 3 2 13

5 10 11 8

9 6 7 12

4 15 14 1

This exactly matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A.

**Expressions**

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

* Variables
* Numbers
* Operators
* Functions

**Variables**

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

num\_students = 25

Creates a 1-by-1 matrix named num\_students and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

**Numbers**

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are

3 -99 0.0001

9.6397238 1.60210e-20 6.02252e23

1i -3.14159j 3e5i

All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10-308 to 10+308.

**Operators**

Expressions use familiar arithmetic operators and precedence rules.

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| \ | Left division (described in "Matrices and Linear Algebra" in Using MATLAB) |
| ^ | Power |
| ' | Complex conjugate transpose |
| ( ) | Specify evaluation order |

**Functions**

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

help elfun

For a list of more advanced mathematical and matrix functions, type

help specfun

help elmat

Some of the functions, like sqrt and sin, are built-in. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in M-files. You can see the code and even modify it if you want. Several special functions provide values of useful constants.

|  |  |
| --- | --- |
| Pi | 3.14159265... |
| i | Imaginary unit, √-1 |
| I | Same as i |
| Eps | Floating-point relative precision, 2-52 |
| Realmin | Smallest floating-point number, 2-1022 |
| Realmax | Largest floating-point number, (2- **ε**)21023 |
| Inf | Infinity |
| NaN | Not-a-number |

**GUI**

A graphical user interface (GUI) is a user interface built with graphical objects, such as buttons, text fields, sliders, and menus. In general, these objects already have meanings to most computer users. For example, when you move a slider, a value changes; when you press an OK button, your settings are applied and the dialog box is dismissed. Of course, to leverage this built-in familiarity, you must be consistent in how you use the various GUI-building components.

Applications that provide GUIs are generally easier to learn and use since the person using the application does not need to know what commands are available or how they work. The action that results from a particular user action can be made clear by the design of the interface.

The sections that follow describe how to create GUIs with MATLAB. This includes laying out the components, programming them to do specific things in response to user actions, and saving and launching the GUI; in other words, the mechanics of creating GUIs. This documentation does not attempt to cover the "art" of good user interface design, which is an entire field unto itself. Topics covered in this section include:

**Creating GUIs with GUIDE**

MATLAB implements GUIs as figure windows containing various styles of uicontrol objects. You must program each object to perform the intended action when activated by the user of the GUI. In addition, you must be able to save and launch your GUI. All of these tasks are simplified by GUIDE, MATLAB's graphical user interface development environment.

**GUI Development Environment**

The process of implementing a GUI involves two basic tasks:

* Laying out the GUI components
* Programming the GUI components

GUIDE primarily is a set of layout tools. However, GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI. This M-file provides a framework for the implementation of the callbacks - the functions that execute when users activate components in the GUI.

**The Implementation of a GUI**

While it is possible to write an M-file that contains all the commands to lay out a GUI, it is easier to use GUIDE to lay out the components interactively and to generate two files that save and launch the GUI:

**A FIG-file** - contains a complete description of the GUI figure and all of its

children (uicontrols and axes), as well as the values of all object properties.

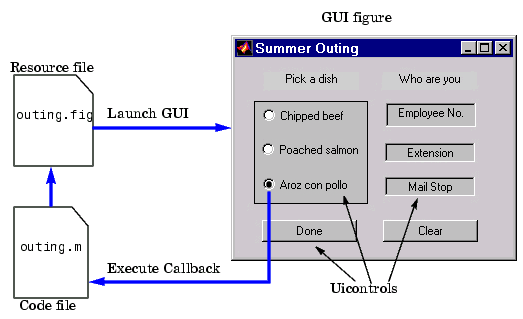
**An M-file** - contains the functions that launch and control the GUI and the

callbacks, which are defined as subfunctions. This M-file is referred to as the

application M-file in this documentation.

Note that the application M-file does not contain the code that lays out the uicontrols; this information is saved in the FIG-file.

The following diagram illustrates the parts of a GUI implementation.



**Features of the GUIDE-Generated Application M-File**

GUIDE simplifies the creation of GUI applications by automatically generating an M-file framework directly from your layout. You can then use this framework to code your application M-file. This approach provides a number of advantages:

The M-file contains code to implement a number of useful features (see Configuring Application Options for information on these features). The M-file adopts an effective approach to managing object handles and executing callback routines (see Creating and Storing the Object Handle Structure for more information). The M-files provides a way to manage global data (see Managing GUI Data for more information).

The automatically inserted subfunction prototypes for callbacks ensure compatibility with future releases. For more information, see Generating Callback Function Prototypes for information on syntax and arguments.

You can elect to have GUIDE generate only the FIG-file and write the application M-file yourself. Keep in mind that there are no uicontrol creation commands in the application M-file; the layout information is contained in the FIG-file generated by the Layout Editor.

**Beginning the Implementation Process**

To begin implementing your GUI, proceed to the following sections:

**Getting Started with GUIDE - the basics of using GUIDE.**

**Selecting GUIDE Application Options** - set both FIG-file and M-file options.

**Using the Layout Editor** - begin laying out the GUI.

**Understanding the Application M-File** - discussion of programming techniques

used in the application M-file.

**Application Examples** - a collection of examples that illustrate techniques

which are useful for implementing GUIs.

**Command-Line Accessibility**

When MATLAB creates a graph, the figure and axes are included in the list of children of their respective parents and their handles are available through commands such as findobj, set, and get. If you issue another plotting command, the output is directed to the current figure and axes.

GUIs are also created in figure windows. Generally, you do not want GUI figures to be available as targets for graphics output, since issuing a plotting command could direct the output to the GUI figure, resulting in the graph appearing in the middle of the GUI.

In contrast, if you create a GUI that contains an axes and you want commands entered in the command window to display in this axes, you should enable command-line access.

**User Interface Controls**

The Layout Editor component palette contains the user interface controls that you can use in your GUI. These components are MATLAB uicontrol objects and are programmable via their Callback properties. This section provides information on these components.

* Push Buttons
* Sliders
* Toggle Buttons
* Frames
* Radio Buttons
* Listboxes
* Checkboxes
* Popup Menus
* Edit Text
* Axes
* Static Text
* Figures

**Push Buttons**

Push buttons generate an action when pressed (e.g., an OK button may close a dialog box and apply settings). When you click down on a push button, it appears depressed; when you release the mouse, the button's appearance returns to its nondepressed state; and its callback executes on the button up event.

**Properties to Set**

**String** - set this property to the character string you want displayed on the push button.

**Tag** - GUIDE uses the Tag property to name the callback subfunction in the application M-file. Set Tag to a descriptive name (e.g., close\_button) before activating the GUI.

**Programming the Callback**

When the user clicks on the push button, its callback executes. Push buttons do not return a value or maintain a state.

**Toggle Buttons**

Toggle buttons generate an action and indicate a binary state (e.g., on or off). When you click on a toggle button, it appears depressed and remains depressed when you release the mouse button, at which point the callback executes. A subsequent mouse click returns the toggle button to the nondepressed state and again executes its callback.

**Programming the Callback**

The callback routine needs to query the toggle button to determine what state it is in. MATLAB sets the Value property equal to the Max property when the toggle button is depressed (Max is 1 by default) and equal to the Min property when the toggle button is not depressed (Min is 0 by default).

**From the GUIDE Application M-File**

The following code illustrates how to program the callback in the GUIDE application M-file.

function varargout = togglebutton1\_Callback(h,eventdata,handles,varargin)

button\_state = get(h,'Value');

if button\_state == get(h,'Max')

% toggle button is pressed

elseif button\_state == get(h,'Min')

% toggle button is not pressed

end

**Adding an Image to a Push Button or Toggle Button**

Assign the CData property an m-by-n-by-3 array of RGB values that define a truecolor image. For example, the array a defines 16-by-128 truecolor image using random values between 0 and 1 (generated by rand).

a(:,:,1) = rand(16,128);

a(:,:,2) = rand(16,128);

a(:,:,3) = rand(16,128);

set(h,'CData',a)

**Radio Buttons**

Radio buttons are similar to checkboxes, but are intended to be mutually exclusive within a group of related radio buttons (i.e., only one button is in a selected state at any given time). To activate a radio button, click the mouse button on the object. The display indicates the state of the button.

**Implementing Mutually Exclusive Behavior**

Radio buttons have two states - selected and not selected. You can query and set the state of a radio button through its Value property:

Value = Max, button is selected.

Value = Min, button is not selected.

To make radio buttons mutually exclusive within a group, the callback for each radio button must set the Value property to 0 on all other radio buttons in the group. MATLAB sets the Value property to 1 on the radio button clicked by the user.

The following subfunction, when added to the application M-file, can be called by each radio button callback. The argument is an array containing the handles of all other radio buttons in the group that must be deselected.

function mutual\_exclude(off)

set(off,'Value',0)

**Obtaining the Radio Button Handles.**

The handles of the radio buttons are available from the handles structure, which contains the handles of all components in the GUI. This structure is an input argument to all radio button callbacks.

The following code shows the call to mutual\_exclude being made from the first radio button's callback in a group of four radio buttons.

function varargout = radiobutton1\_Callback(h,eventdata,handles,varargin)

off = [handles.radiobutton2,handles.radiobutton3,handles.radiobutton4];

mutual\_exclude(off)

% Continue with callback

.

.

.

After setting the radio buttons to the appropriate state, the callback can continue with its implementation-specific tasks.

**Checkboxes**

Check boxes generate an action when clicked and indicate their state as checked or not checked. Check boxes are useful when providing the user with a number of independent choices that set a mode (e.g., display a toolbar or generate callback function prototypes).

The Value property indicates the state of the check box by taking on the value of the Max or Min property (1 and 0 respectively by default):

Value = Max, box is checked.

Value = Min, box is not checked.

You can determine the current state of a check box from within its callback by querying the state of its Value property, as illustrated in the following example:

function checkbox1\_Callback(h,eventdata,handles,varargin)

if (get(h,'Value') == get(h,'Max'))

% then checkbox is checked-take approriate action

else

% checkbox is not checked-take approriate action

end

**Edit Text**

Edit text controls are fields that enable users to enter or modify text strings. Use edit text when you want text as input. The String property contains the text entered by the user.

To obtain the string typed by the user, get the String property in the callback.

function edittext1\_Callback(h,eventdata, handles,varargin)

user\_string = get(h,'string');

% proceed with callback...

**Obtaining Numeric Data from an Edit Test Component**

MATLAB returns the value of the edit text String property as a character string. If you want users to enter numeric values, you must convert the characters to numbers. You can do this using the str2double command, which converts strings to doubles. If the user enters non-numeric characters, str2double returns NaN.

You can use the following code in the edit text callback. It gets the value of the String property and converts it to a double. It then checks if the converted value is NaN, indicating the user entered a non-numeric character (isnan) and displays an error dialog (errordlg).

function edittext1\_Callback(h,eventdata,handles,varargin)

user\_entry = str2double(get(h,'string'));

if isnan(user\_entry)

errordlg('You must enter a numeric value','Bad Input','modal')

end

% proceed with callback...

**Triggering Callback Execution**

On UNIX systems, clicking on the menubar of the figure window causes the edit text callback to execute. However, on Microsoft Windows systems, if an editable text box has focus, clicking on the menubar does not cause the editable text callback routine to execute. This behavior is consistent with the respective platform conventions. Clicking on other components in the GUI execute the callback.

**Static Text**

Static text controls displays lines of text. Static text is typically used to label other controls, provide directions to the user, or indicate values associated with a slider. Users cannot change static text interactively and there is no way to invoke the callback routine associated with it.

**Frames**

Frames are boxes that enclose regions of a figure window. Frames can make a user interface easier to understand by visually grouping related controls. Frames have no callback routines associated with them and only uicontrols can appear within frames (axes cannot).

**Placing Components on Top of Frames**

Frames are opaque. If you add a frame after adding components that you want to be positioned within the frame, you need to bring forward those components. Use the Bring to Front and Send to Back operations in the Layout menu for this purpose.

**List Boxes**

List boxes display a list of items and enable users to select one or more items.

The String property contains the list of strings displayed in the list box. The first item in the list has an index of 1.

The Value property contains the index into the list of strings that correspond to the selected item. If the user selects multiple items, then Value is a vector of indices.

By default, the first item in the list is highlighted when the list box is first displayed. If you do not want any item highlighted, then set the Value property to empty, [].

The ListboxTop property defines which string in the list displays as the top most item when the list box is not large enough to display all list entries. ListboxTop is an index into the array of strings defined by the String property and must have a value between 1 and the number of strings. Noninteger values are fixed to the next lowest integer.

**Single or Multiple Selection**

The values of the Min and Max properties determine whether users can make single or multiple selections:

If Max - Min > 1, then list boxes allow multiple item selection.

If Max - Min <= 1, then list boxes do not allow multiple item selection.

**Selection Type**

Listboxes differentiate between single and double clicks on an item and set the figure SelectionType property to normal or open accordingly. See Triggering Callback Execution for information on how to program multiple selection.

**Triggering Callback Execution**

MATLAB evaluates the list box's callback after the mouse button is released or a keypress event (including arrow keys) that changes the Value property (i.e., any time the user clicks on an item, but not when clicking on the list box scrollbar). This means the callback is executed after the first click of a double-click on a single item or when the user is making multiple selections.

In these situations, you need to add another component, such as a Done button (push button) and program its callback routine to query the list box Value property (and possibly the figure SelectionType property) instead of creating a callback for the list box. If you are using the automatically generated application M-file option, you need to either:

Set the list box Callback property to the empty string ('') and remove the callback subfunction from the application M-file. Leave the callback subfunction stub in the application M-file so that no code executes when users click on list box items.

The first choice is best if you are sure you will not use the list box callback and you want to minimize the size and efficiency of the application M-file. However, if you think you may want to define a callback for the list box at some time, it is simpler to leave the callback stub in the M-file.

**Popup Menus**

Popup menus open to display a list of choices when users press the arrow.

The String property contains the list of string displayed in the popup menu. The Value property contains the index into the list of strings that correspond to the selected item.

When not open, a popup menu displays the current choice, which is determined by the index contained in the Value property. The first item in the list has an index of 1.

Popup menus are useful when you want to provide users with a number of mutually exclusive choices, but do not want to take up the amount of space that a series of radio buttons requires.

**Programming the Popup Menu**

You can program the popup menu callback to work by checking only the index of the item selected (contained in the Value property) or you can obtain the actual string contained in the selected item.

This callback checks the index of the selected item and uses a switch statement to take action based on the value. If the contents of the popup menu is fixed, then you can use this approach.

function varargout = popupmenu1\_Callback(h,eventdata,handles,varargin)

val = get(h,'Value');

switch val

case 1

% The user selected the first item

case 2

% The user selected the second item

% etc.

This callback obtains the actual string selected in the popup menu. It uses the value to index into the list of strings. This approach may be useful if your program dynamically loads the contents of the popup menu based on user action and you need to obtain the selected string. Note that it is necessary to convert the value returned by the String property from a cell array to a string.

function varargout = popupmenu1\_Callback(h,eventdata,handles,varargin)

val = get(h,'Value');

string\_list = get(h,'String');

selected\_string = string\_list{val}; % convert from cell array to string

% etc.

**Enabling or Disabling Controls**

You can control whether a control responds to mouse button clicks by setting the Enable property. Controls have three states:

on - The control is operational

off - The control is disabled and its label (set by the string property) is

grayed out.

inactive - The control is disabled, but its label is not grayed out.

When a control is disabled, clicking on it with the left mouse button does not execute its callback routine. However, the left-click causes two other callback routines to execute:

First the figure WindowButtonDownFcn callback executes. Then the control's ButtonDownFcn callback executes.

A right mouse button click on a disabled control posts a context menu, if one is defined for that control. See the Enable property description for more details.

**Axes**

Axes enable your GUI to display graphics (e.g., graphs and images). Like all graphics objects, axes have properties that you can set to control many aspects of its behavior and appearance. See Axes Properties for general information on axes objects.

**Axes Callbacks**

Axes are not uicontrol objects, but can be programmed to execute a callback when users click a mouse button in the axes. Use the axes ButtonDownFcn property to define the callback.

**Plotting to Axes in GUIs**

GUIs that contain axes should ensure the Command-line accessibility option in the Application Options dialog is set to Callback (the default). This enables you to issue plotting commands from callbacks without explicitly specifying the target axes.

**GUIs with Multiple Axes**

If a GUI has multiple axes, you should explicitly specify which axes you want to target when you issue plotting commands. You can do this using the axes command and the handles structure. For example,

axes(handles.axes1)

makes the axes whose Tag property is axes1 the current axes, and therefore the target for plotting commands. You can switch the current axes whenever you want to target a different axes. See GUI with Multiple Axes for and example that uses two axes.

**Figure**

Figures are the windows that contain the GUI you design with the Layout Editor. See the description of figure properties for information on what figure characteristics you can control.