Introduction to Graphical User Interface (GUI) MATLAB 6.5

Presented By:

Refaat Yousef Al Ashi 199901469
Ahmed Al Ameri 199900378

Coordinated By:

Prof. Abdulla Ismail Abdulla
Introduction

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button. This chapter introduces the basic elements of the MATLAB GUIs. The chapter does not contain a complete description of components or GUI features, but it does provide the basics required to create functional GUIs for your programs.

1.1 How a Graphical User Interface Works

A graphical user interface provides the user with a familiar environment in which to work. This environment contains pushbuttons, toggle buttons, lists, menus, text boxes, and so forth, all of which are already familiar to the user, so that he or she can concentrate on using the application rather than on the mechanics involved in doing things. However, GUIs are harder for the programmer because a GUI-based program must be prepared for mouse clicks (or possibly keyboard input) for any GUI element at any time. Such inputs are known as events, and a program that responds to events is said to be event driven. The three principal elements required to create a MATLAB Graphical User Interface are

1. Components. Each item on a MATLAB GUI (pushbuttons, labels, edit boxes, etc.) is a graphical component. The types of components include graphical controls (pushbuttons, edit boxes, lists, sliders, etc.), static elements (frames and text strings), menus, and axes. Graphical controls and static elements are created by the function uicontrol, and menus are created by the functions uimenu and uicontextmenu. Axes, which are used to display graphical data, are created by the function axes.

2. Figures. The components of a GUI must be arranged within a figure, which is a window on the computer screen. In the past, figures have been created automatically whenever we have plotted data. However, empty figures can be created with the function figure and can be used to hold any combination of components.

3. Callbacks. Finally, there must be some way to perform an action if a user clicks a mouse on a button or types information on a keyboard. A mouse click or a key press is an event, and the MATLAB program must respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB code that implements the function of the button to be executed. The code executed in response to an event is known as a call back. There must be a callback to implement the function of each graphical component on the GUI. The basic GUI elements are summarized in Table 1.1, and sample elements are shown in Figure 1.1. We will be studying examples of these elements and then build working GUIs from them.
1.2 Creating and Displaying a Graphical User Interface

MATLAB GUIs are created using a tool called guide, the GUI Development Environment. This tool allows a programmer to layout the GUI, selecting and aligning the GUI components to be placed in it. Once the components are in place, the programmer can edit their properties: name, color, size, font, text to display, and so forth. When guide saves the GUI, it creates working program including skeleton functions that the programmer can modify to implement the behavior of the GUI. When guide is executed, it creates the Layout Editor, shown in Figure 1.2. The large white area with grid lines is the layout area, where a programmer can layout the GUI. The Layout Editor window has a palate of GUI components along the left side of the layout area. A user can create any number of GUI components by first clicking on the desired component, and then dragging its outline in the layout area. The top of the window has a toolbar with a series of useful tools that allow the user to distribute and align GUI components, modify the properties of GUI components, add menus to GUIs, and so on. The basic steps required to create a MATLAB GUI are:

1. Decide what elements are required for the GUI and what the function of each element will be. Make a rough layout of the components by hand on a piece of paper.
Table 1.1 Some Basic GUI Components

<table>
<thead>
<tr>
<th>Element</th>
<th>Created By</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical Controls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pushbutton</td>
<td>uicontrol</td>
<td>A graphical component that implements a pushbutton. It triggers a callback when clicked with a mouse.</td>
</tr>
<tr>
<td>Toggle button</td>
<td>uicontrol</td>
<td>A graphical component that implements a toggle button. A toggle button is either “on” or “off,” and it changes state each time that it is clicked. Each mouse button click also triggers a callback.</td>
</tr>
<tr>
<td>Radio button</td>
<td>uicontrol</td>
<td>A radio button is a type of toggle button that appears as a small circle with a dot in the middle when it is “on.” Groups of radio buttons are used to implement mutually exclusive choices. Each mouse click on a radio button triggers a callback.</td>
</tr>
<tr>
<td>Check box</td>
<td>uicontrol</td>
<td>A check box is a type of toggle button that appears as a small square with a check mark in it when it is “on.” Each mouse click on a check box triggers a callback.</td>
</tr>
<tr>
<td>Edit box</td>
<td>uicontrol</td>
<td>An edit box displays a text string and allows the user to modify the information displayed. A callback is triggered when the user presses the Enter key.</td>
</tr>
<tr>
<td>List box</td>
<td>uicontrol</td>
<td>A list box is a graphical control that displays a series of text strings. A user can select one of the text strings by single- or double-clicking on it. A callback is triggered when the user selects a string.</td>
</tr>
<tr>
<td>Popup menus</td>
<td>uicontrol</td>
<td>A popup menu is a graphical control that displays a series of text strings in response to a mouse click. When the popup menu is not clicked on, only the currently selected string is visible.</td>
</tr>
<tr>
<td>Slider</td>
<td>uicontrol</td>
<td>A slider is a graphical control to adjust a value in a smooth, continuous fashion by dragging the control with a mouse. Each slider change triggers a callback.</td>
</tr>
<tr>
<td>Static Elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td>uicontrol</td>
<td>Creates a frame, which is a rectangular box within a figure. Frames are used to group sets of controls together. Frames never trigger callbacks.</td>
</tr>
<tr>
<td>Text field</td>
<td>uicontrol</td>
<td>Creates a label, which is a text string located at a point on the figure. Text fields never trigger callbacks.</td>
</tr>
<tr>
<td>Menus and Axes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menu items</td>
<td>uimenu</td>
<td>Creates a menu item. Menu items trigger a callback when a mouse button is released over them.</td>
</tr>
<tr>
<td>Context menus</td>
<td>uicontextmenu</td>
<td>Creates a context menu, which is a menu that appears over a graphical object when a user right-clicks the mouse on that object.</td>
</tr>
<tr>
<td>Axes</td>
<td>axes</td>
<td>Creates a new set of axes to display data on. Axes never trigger callbacks.</td>
</tr>
</tbody>
</table>

Table 1.1 Some Basic GUI Components
Figure 1.1 A Figure Window showing examples of MATLAB GUI elements. From top to bottom and left to right, the elements are: (1) a pushbutton; (2) a toggle button in the 'on' state; (3) two radio buttons surrounded by a frame; (4) a check box; (5) a text field and an edit box; (6) a slider; (7) a set of axes; and (8) a list box.

2. Use a MATLAB tool called guide (GUI Development Environment) to layout the Components on a figure. The size of the figure and the alignment and spacing of components on the figure can be adjusted using the tools built into guide.

3. Use a MATLAB tool called the Property Inspector (built into guide) to give each component a name (a "tag") and to set the characteristics of each component, such as its color, the text it displays, and so on.

4. Save the figure to a file. When the figure is saved, two files will be created on disk with the same name but different extents. The fig file contains the actual GUI that you have created, and the M-file contains the code to load the figure and skeleton call backs for each GUI element.

5. Write code to implement the behavior associated with each callback function.
As an example of these steps, let's consider a simple GUI that contains a single pushbutton and a single text string. Each time that the pushbutton is clicked, the text string will be updated to show the total number of clicks since the GUI started.

Step 1: The design of this GM is very simple. It contains a single pushbutton and a single text field. The callback from the pushbutton will cause the number displayed in the text field to increase by one each time that the button is pressed. A rough sketch of the GUI is shown in Figure 1.3.
Step 2: To layout the components on the GUI, run the MATLAB function guide. When guide is executed, it creates the window shown in Figure 1.2.

![Guide Window](image)

Figure 1.4 The completed GUI layout within the guide window

First, we must set the size of the layout area, which will become the size the final GUI. We do this by dragging the small square on the lower right corner of the layout area until it has the desired size and shape. Then, click on the "pushbutton" button in the list of GUI components, and create the shape of the pushbutton in the layout area. Finally, click on the "text" button in the list GUI components, and create the shape of the text field in the layout area. The resulting figure after these steps is shown in Figure 1.4. We could now adjust the alignment of these two elements using the Alignment Tool, if desired.

Step 3: To set the properties of the pushbutton, click on the button in the layout area and then select "Property Inspector" from the toolbar. Alternatively, right-click on the button and select "Inspect Properties" from the popup menu. The Property Inspector window shown in Figure 1.5 will appear. Note this window lists every property available for the pushbutton and allows us set each value using a GUI interface. The Property Inspector performs the same function as the get and set functions, but in a much more convenient form.
Figure 1.5 The Property Inspector showing the properties of the pushbutton. Note that the String is set to 'Click Here', and the Tag is set to 'MyFirstButton'.

For the pushbutton, we may set many properties such as color, size, font, text alignment, and so on. However, we must set two properties: the String property, which contains the text to be displayed, and the Tag property, which is the name of the pushbutton. In this case, the String property will be set to 'click Here', and the Tag property will be set to MyFirstButton. For the text field, we must set two properties: the String property, which contains the text to be displayed, and the Tag property, which is the name of the text field. This name will be needed by the callback function to locate and update the text field. In this case, the String property will be set to 'Total clicks: 0', and the Tag property defaulted to 'MyFirstText'. The layout area after these steps is shown in Figure 1.6. It is possible to set the properties of the figure itself by clicking on a clear spot in the Layout Editor, and then using the Property Inspector to examine and set the figure's properties. Although not required, it is a good idea to set the figure's Name property. The string in the Name property will be displayed in the title bar of the resulting GUI when it is executed.
Figure 1.6 The design area after the properties of the pushbutton and the text field have been modified.

Step 4: We will now save the layout area under the name MyFirstGUI. Select the "File/SaveAs" menu item, type the name MyFirstGUI as the file name, and click "Save". This action will automatically create two files, MyFirstGUI.fig and MyFirstGUI.m. The figure file contains the actual GUI that we have created. The M-file contains code that loads the figure file and creates the GUI, plus a skeleton callback function for each active GUI component.

At this point, we have a complete Gm, but one that does not yet do the job it was designed to do. You can start this Gm by typing MyFirstGUI in the Command Window, as shown in Figure 1.7. If the button is clicked on this GUI, the following message will appear in the Command Window: MyFirstButton Callback not implemented yet. A portion of the M-file automatically created by guide is shown in Figure 1.8. This file contains function MyFirstGUI, plus dummy sub functions implementing the callbacks for each active GUI component. If function MyFirstGUI is called without arguments, then the function displays the Gm contained in file
Figure 1.7 Typing MyFirstGUI in the Command Window starts the GUI.

MyFirstGUI.fig. If function MyFirstGUI is called with arguments, then the function assumes that the first arguments the name of a sub function, and it calls that function using feval, passing the other arguments on to that function. Each callback function handles events from a single GUI component. If a mouse click (or keyboard input for Edit Fields) occurs on the GUI component, then the component's callback function will be automatically called by MATLAB. The name of the callback function will be the value in the Tag property of the GUI component plus the characters "_Callback". Thus, the callback function for MyFirstButton will be named MyFirstButton_Callback. M-files created by guide contain callbacks for each active GUI component, but these callbacks simply display a message saying that the function of the callback has not been implemented yet.

Step 5: Now, we need to implement the callback sub function for the pushbutton. This function will include a persistent variable that can be used to count the number of clicks that have occurred. When a click occurs on the pushbutton, MATLAB will call the function MyFirstGUI with MyFirstButton_callback as the first argument. Then function MyFirstGUI will call sub function MyFirstButton_callback, as shown in Figure 1.9. This function should increase the count of clicks by one, create a new text string containing the count, and store the new string in the String property of the text field MyFirstText.
Figure 1.8 The M-file for MyFirstGUI, automatically created by guide.

A function to perform this step is shown below:

```matlab
function varargout = MyFirstGUI(varargin)
    % MyFirstGUI Application M-file for MyFirstGUI.fig
    % FIG = MyFirstGUI launch MyFirstGUI GUI.
    % MyFirstGUI('callback_name', ...) invoke the named callback.

    % Last Modified by GUIDE v2.0 22-Jun-2001 21:14:46
    if nargin == 0 % LAUNCH GUI
        fig = openfig(fullfile('MyFirstGUI.fig', 'reuse')); % If called without an
        % argument, open the GUI.
        % Use system color scheme for figure:
        set(fig, 'Color', get(0, 'defaultUicontrolBackgroundColor'))
        % Generate a structure of handles to pass to callbacks, and store it.
        handles = guihandles(fig);
        guidata(fig, handles);
        if nargin > 0
            varargout{1} = fig;
        end
    elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK
        try
            varargout{1:nargin} = feval(varargin{:}); % FEVAL switchyard
        catch
            disp(lasterr);
        end
    end

    % Callbacks are subfunctions.

    function varargout = MyFirstButton_Callback(h, eventdata, handles, varargin)
    % Stub for Callback of the uicontrol handles.MyFirstButton.
    disp('MyFirstButton Callback not implemented yet.')
```

% Declare and initialize variable to store the count persistent count
if isempty(count)
    count = 0;
end
% Update count
count = count + 1;

% Create new string
str = sprintf('Total Clicks: %d', count);

% Update the text field
set (handles.MyFirstText, 'String', str);

Figure 1.9 Event handling in program MyFirstGUI. When a user clicks on the button with the mouse, the function MyFirstGUI is called automatically with the argument MyFirstButton_callback. Function MyFirstGUI in turn calls sub function
MyFirstButton_Callback. This function increments count, and then saves the new count in the text field on the GUI.

Figure 1.10 The resulting program after three button pushes.

Note that this function declares a persistent variable count and initializes it to zero. Each time that the function is called, it increments count by 1 and creates a new string containing the count. Then, the function updates the string displayed in the text field MyFirstText. The resulting program is executed by typing MyFirstGUI in the Command Window. When the user clicks on the button, MATLAB automatically calls function MyFirstGUI with MYFirstButton_Callback as the first argument, and function MyFirstGUI calls sub function MyFirstButton_Callback. This function increments variable count by one and updates the value displayed in the text field. The resulting GUI after three button pushes is shown in Figure 1.10.

**Good Programming Practice**

Use guide to layout a new GUI, and use the Property Inspector to set the initial properties of each component such as the text displayed on the component, the color of the component, and the name of the callback function, if required. After creating a GUI with guide, manually edit the resulting function to add comments describing its purpose and components and to implement the function of callbacks.

### 1.2.1 A Look Under the Hood

Figure 1.8 shows the M-file that was automatically generated by guide for MyFirstGUI. We will now examine this M-file more closely to understand how it works.

First, let's look at the function declaration itself. Note that this function uses varargin to represent its input arguments and varargout to represent its output results. Function varargin can represent an arbitrary number of input arguments, and function varargout can represent a varying number of output arguments. Therefore, a user can call function MyFirstGUI with any number of arguments.

**Calling the M-File without Arguments**

If the user calls MyFirstGUI without arguments, the value returned by margin will be zero. In this case, the program loads the Gm from the figure file MyFirstGUI.fig using the openfig function. The form of this function is
fig = openfig(mfilename, 'reuse');

where mfilename is the name of the figure file to load. The second argument in the function specifies whether there can be only one copy of the figure running at a given time, or whether multiple copies can be run. If the argument is 'reuse', then only one copy of the figure can be run. If openfig is called with the 'reuse' option and the specified figure already exists, the preexisting figure will be brought to the top of the screen and reused. In contrast, if the argument is 'new', multiple copies of the figure can be run. If openfig is called with the 'new' option, a new copy of the figure will be created each time. By default, a GUI created by guide has the 'reuse' option, so only one copy of the figure can exist at any time. If you want to have multiple copies of the GUI, you must manually edit this function call. After the figure is loaded, the function executes the statement

```matlab
set(fig, 'Color', get(0, 'defaultUicontrolBackgroundColor'));
```

This function sets the background color of the figure to match the default background color used by the computer on which MATLAB is executing. This function makes the color of the GUI match the color of native windows on the computer. Therefore, a GUI can be written on a Windows-based PC and used on a UNIX-based computer, and vice versa. It will look natural in either environment. The next two statements create a structure containing the handles of all the objects in the current figure and store that structure as application data in the figure.

```matlab
handles = guihandles(fig); guidata(fig, handles);
```

Function guihandles creates a structure containing handles to all of the objects within the specified figure. The element names in the structure correspond to the Tag properties of each GUI component, and the values are the handles of each component. For example, the handle structure returned in MyFirstGUI.m is

```matlab
>> handles = guihandles(fig)
handles =
  figure1: 99.0005
  MyFirstText: 3.0021
  MyFirstButton: 100.0007
```

There are three GUI components in this figure the figure itself, plus a text field and a pushbutton. Function guidata saves the handles structure as application data in the figure, using the setappdata function. The final set of statements here returns the figure handle to the caller if an output argument was specified in the call to MyFirstGUI.
Calling the M-File with Arguments
If the user calls MyFirstGUI with arguments, the value returned by nargin will be greater than zero. In this case, the program treats the first calling argument as a callback function name and executes the function using feval. This function executes the function named in varargin{1} and passes all of the remaining arguments (varargin{2}, varargin{3}, etc.) to the function. This mechanism allows callback functions to be sub functions that cannot be accidentally called from some other part of the program.

1.2.2 The Structure of a Call back Sub function
Every callback sub function has the standard form

```matlab
function vararginout = ComponentTag_Callback(h, eventdata, ...
    handles, varargin)
```

where ComponentTag is the name of the component generating the callback (the string in its Tag property). The arguments of this sub function are

- `h` - The handle of the parent figure
- `eventdata` - A currently unused (in MATLAB Version 6) array.
- `handles` - The handles structure contains the handles of all GUI components on the figure.
- `varargin` - A supplemental argument passing any additional calling arguments to the callback function. A programmer can use this feature to provide additional information to the callback function if needed.

Note that each callback function has full access to the handles structure, so each callback function can modify any GUI component in the figure. We took advantage of this structure in the callback function for the pushbutton in MyFirstGUI, where the callback function for the pushbutton modified the text displayed in the text field.

```matlab
set (handles.MyFirstText,'String',str);
```

1.2.3 Adding Application Data to a Figure
It is possible to store application-specific information needed by a GUI program in the handles structure instead of using global or persistent memory for that data. The resulting GUI design is more robust because other MATLAB programs cannot accidentally modify the global GUI data and because multiple copies of the same GUI cannot interfere with each other.
To add local data to the handles structure, we must manually modify the Mfile after it is created by guide. A programmer adds the application data to the handles structure after the call to guidata and before the call to guidata. For example, to add the number of mouse clicks count to the handles structure, we would modify the program as follows:

```matlab
% Generate a structure of handles to pass to callbacks
handles = guihandles(fig);

% Add count to the structure.
handles.count = 0;

% Store the structure
guidata(fig, handles);
```

This application data will now be passed with the handles structure to every callback function, where it can be used. A version of the pushbutton callback function that uses the count value in the handles data structure is shown below. Note that we must save the handles structure with a call to guidata if any of the information in it has been modified.

```matlab
function varargout = MyFirstButton_Callback(hObject, eventdata, ...
    handles, varargin)

    % Update count
    handles.count = handles.count + 1;

    % Save the updated handles structure
    guidata(hObject, handles);

    % Create new string
    str = sprintf('%i Total Clicks! %d', handles.count);

    % Update the text field
    set(handles.MyFirstText, 'String', str);
```

**Good Programming Practice**

Store GUI application data in the handles structure, so that it will automatically be available to any callback function.

If you modify any of the GUI application data in the handles structure, be sure to save the structure with a call to guidata before exiting the function where the modifications occurred.

**1.2.4 A Few Useful Functions**

Three special functions are used occasionally in the design of callback functions: gcbo, gcbf, and findobj. Although these functions are less needed with MATLAB 6.5 GUIs than with earlier versions, they are still very useful, and a programmer is sure to encounter them.

Function gcbo (get callbackobject) returns the handle of the object that generated the callback, and function gcbf (get callbackfigure) returns the handle of the figure containing that object. These functions can be used by the callback function to determine the object and figure producing the callback, so that it can modify objects on that figure.
Function `findobj` searches through all of the child objects within a parent object, looking for ones that have a specific value of a specified property. It returns a handle to any objects with the matching characteristics. The most common form of `findobj` is

```matlab
Handle = findobj(parent,'Property','Value');
```

where `parent` is the handle of a parent object such as a figure, 'Property' is the property to examine, and 'Value' is the value to look for. For example, suppose that a programmer would like to change the text on a pushbutton with the tag 'Button1' when a callback function executes. The programmer could find the pushbutton and replace the text with the following statements

```matlab
Handle = findobj(gcf,'Tag','Button1');
set(Handle,'String','New text');
```

### 1.3 Object Properties

Every GUI object includes an extensive list of properties that can be used to customize the object. These properties are slightly different for each type of object (figures, axes, uicontrols, etc.). All of the properties for all types of objects are documented on the online Help Browser, but a few of the more important properties for figure and uicontrol objects are summarized in Tables 1.2 and 1.3.

Object properties may be modified using either the Property Inspector or the `get` and `set` functions. Although the Property Inspector is a convenient way to adjust properties during GUI design, we must use `get` and `set` to adjust them dynamically from within a program, such as in a callback function.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>Specifies the color of the figure. The value is either a predefined color such as 'r', 'g', or 'b', or a 3-element vector specifying the red, green, and blue components of the color on a 0-1 scale. For example, the color magenta would be specified by [0 0 1].</td>
</tr>
<tr>
<td>MenuBar</td>
<td>Specifies whether or not the default set of menus appears on the figure. Possible values are 'figure' to display the default menus, or 'none' to delete them.</td>
</tr>
<tr>
<td>Name</td>
<td>A string containing the name that appears in the title bar of a figure.</td>
</tr>
<tr>
<td>NumberTitle</td>
<td>Specifies whether or not the figure number appears in the title bar. Possible values are 'on' or 'off'.</td>
</tr>
<tr>
<td>Position</td>
<td>Specifies the position of a figure on the screen, in the units specified by the 'units' property. This value accepts a 4-element vector in which the first two elements are the x and y positions at the lower-left hand corner of the figure, and the next two elements are the width and height of the figure.</td>
</tr>
<tr>
<td>SelectionType</td>
<td>Specifies the type of selection for the last mouse click on this figure. A single click returns type 'normal'. A double click returns type 'open'. There are additional options; see the MATLAB on-line documentation.</td>
</tr>
<tr>
<td>Tag</td>
<td>The 'name' of the figure, which can be used to locate it.</td>
</tr>
<tr>
<td>Units</td>
<td>The units used to describe the position of the figure. Possible choices are 'inches', 'centimeters', 'normalized', 'points', 'pixels', or 'characters'. The default units are 'pixels'.</td>
</tr>
<tr>
<td>Visible</td>
<td>Specifies whether or not this figure is visible. Possible values are 'on' or 'off'.</td>
</tr>
<tr>
<td>WindowStyle</td>
<td>Specifies whether this figure is normal or modal (see discussion of Dialog Boxes). Possible values are 'normal' or 'modal'.</td>
</tr>
</tbody>
</table>

Table 1.2 Important figure Properties
1.4 Graphical User Interface Components

This section summarizes the basic characteristics of common graphical user interface components. It describes how to create and use each component, as well as the types of events each component can generate. The components discussed in this section are

- Text Fields
- Edit Boxes
- Frames
- Pushbuttons
- Toggle Buttons
- Checkboxes
- Radio Buttons
- Popup Menus
- List Boxes
- Slide
1.4.1 Text Fields
A text-field is a graphical object that displays a text string. You can specify how the text is aligned in the display area by setting the horizontal alignment property. By default, text fields are horizontally centered. A text field is created by creating a uicontrol whose style property is 'edit'. A text field may be added to a GUI by using the text tool in the Layout Editor. Text fields do not create callbacks, but the value displayed in the text field can be updated in a callback function by changing the text field's String property, as shown in Section 1.2.

1.4.2 Edit Boxes
An edit box is a graphical object that allows a user to enter a text string. The edit box generates a callback when the user presses the Enter key after typing a string into the box. An edit box is created by creating a uicontrol whose style property is 'edit'. An edit box may be added to a GUI by using the edit box tool in the Layout Editor. Figure 1.11a shows a simple GUI containing an edit box named 'Edit Box' and a text field named 'TextBox'. When a user types a string into the edit box, it automatically calls the function EditBox_Callback, which is shown in Figure 1.11b. This function locates the edit box using the handles structure and recovers the string typed by the user. Then, it
locates the text field and displays the string in the text field. Figure 1.12 shows this GUI just after it has started and after the user has typed the word "Hello" in the edit box.

1.4.3 Frames
A frame is a graphical object that displays a rectangle on the GUI. You can use frames to draw boxes around groups of logically related objects. For example, a frame is used to group the radio buttons together on Figure 1.1. A frame is created by creating a uicontrol whose style property is 'frame'. A frame maybe added to a GUI by using the frame tool in the LayoutEditor. Frames do not generate callbacks.

1.4.4 Pushbuttons
A pushbutton is a component that a user can click on to trigger a specific action. The pushbutton generates a callback when the user clicks the mouse on it. A pushbutton is created by creating a uicontrol whose style property is 'pushbutton'. A pushbutton may be added to a GUI by using the pushbutton tool in the Layout Editor. Function MyFirstGUI in Figure 1.10 illustrated the use of pushbuttons.

Figure 1.11 (a) Layout of a simple GUI with an edit box and a text field. (b) The callback functions for this GUI.
1.4.5 Toggle Buttons
A toggle button is a type of button that has two states: on (depressed) and off (not depressed). A toggle button switches between these two states whenever the mouse clicks on it, and it generates a callback each time. The 'Value' property of the toggle button is set to max (usually 1) when the button is on, and min (usually 0) when the button is off. A toggle button is created by creating a uicontrol whose style property is toggle button. A toggle button may be added to a GUI by using the toggle button tool in the Layout Editor. Figure 1.13a shows a simple GUI containing a toggle button named 'ToggleButton' and a text field named 'TextBox'. When a user clicks on the toggle button, it automatically calls the function ToggleButton_Callback, which is shown in Figure 1.13b. This function locates the toggle button using the handles structure and recovers its state from the 'Value' property. Then, the function locates the text field and displays the state in the text field. Figure 1.14 shows this GUI just after it has started, and after the user has clicked on the toggle button for the first time.

1.4.6 Checkboxes and Radio Buttons
Checkboxes and radio buttons are essentially identical to toggle buttons except that they have different shapes. Like toggle buttons, checkboxes and radio buttons have two states: on and off. They switch between these two states whenever the mouse clicks on them, generating a callback each time. The 'Value' property of the checkbox or radio button is set to max (usually 1) when they are on, and min (usually 0) when they are off. Both checkboxes and radio buttons are illustrated in Figure 1.1.
A checkbox is created by creating a uicontrol whose style property is 'checkbox', and a radio button is created by creating a uicontrol whose style property is 'radiobutton'. A checkbox may be added to a GUI by using the checkbox tool in the Layout Editor, and a radio button may be added to a GUI by using the radio button tool in the Layout Editor. Checkboxes are traditionally used to display on/off options, and groups of radio buttons are traditionally used to select among mutually exclusive options. Figure 1.15a shows an example of how to create a group of mutually exclusive options with radio buttons. The GUI in this figure creates three radio buttons, labeled "Option 1," "Option 2," and "Option 3." Each radio button uses the same callback function, but with a separate parameter. The corresponding callback functions are shown in Figure 1.15b. When the user clicks on a radio button, the corresponding callback function is executed. That
function sets the text box to display the current option, turns on that radio button, and turns off all other radio buttons. Note that the GUI uses a frame to group the radio buttons together, making it obvious that they are a set. Figure 1.16 shows this GUI after Option 2 has been selected.

```
function varargout = ToggleButton_Callback(hObject, eventdata, ... 
    handles, varargin)

goto {state = get(handles.ToggleButton, 'Value');
    if state == 0
        set (handles.TextBox, 'String', 'off');
    else
        set (handles.TextBox, 'String', 'On');
    end
```

Figure 1.13 (a) Layout of a simple GUI with a toggle button and a text field. (b) The callback function for this GUI.

Figure 1.14 (a) The GUI produced by program test_togglebutton when the toggle button is off. (b) The GUI when the toggle button is on.
Figure 1.15 (a) Layout of a simple GUI with three radio buttons in a frame, plus a text field to display the current selection. (b) The callback functions for this GUI.

Figure 1.16 The GUI produced by program test_radiobutton

1.4.7 Popup Menus
Popup menus are graphical objects that allow a user to select one of a mutually exclusive list of options. The list of options that the user can select among is specified by a cell array of strings, and the 'Value' property indicates which of the strings is currently selected. A popup menu may be added to a GUI by using the popup menu tool in the Layout Editor.

Figure 1.17a shows an example of a popup menu. The GUI in this figure creates a popup menu with five options, labeled "Option 1," "Option 2," and so forth. The corresponding callback function is shown in Figure 1.17b. The call back function recovers the selected option by checking the 'Value' parameter of the popup menu, and creates and displays a
string containing that value in the text field. Figure 1.18 shows this Gm after Option 4 has been selected.

1.4.8 List Boxes
List boxes are graphical objects that display many lines of text and allow a user to select one or more of those lines. If there are more lines of text than can fit in the list box, a scroll bar will be created to allow the user to scroll up and down within the list box. The lines of text that the user can select among are specified by a cell array of strings, and the' Value' property indicates which of the strings are currently selected. A list box is created by creating a uicontrol whose style property is 'listbox'. A list box may be added to a GUI by using the listbox tool in the Layout Editor.

List boxes can be used to select a single item from a selection of possible choices. In normal GUI usage, a single mouse click on a list item selects that item but does not cause an action to occur. Instead, the action waits on some external trigger, such as a pushbutton. However, a mouse double-click causes an action to happen immediately. Single-click and double-click events can be distinguished using the SelectionType property of the figure in which the clicks occurred. A single mouse click will place the string 'normal' in the SelectionType property, and a double mouse click will place the string 'open' in the SelectionType property.

![Figure 1.17 (a) Layout of a simple GUI with a popup menu and a text field to display the current selection. (b) The callback functions for this GUI.](image)

**Figure 1.17 (a)** Layout of a simple GUI with a popup menu and a text field to display the current selection. (b) The callback functions for this GUI.

![Figure 1.18 The GUI produced by program test_popup.](image)

**Figure 1.18** The GUI produced by program test_popup.
It is also possible for a list box to allow multiple selections from the list. If the difference between the max and min properties of the list box is greater than one, then multiple selections is allowed. Otherwise, only one item may be selected from the list.

Figure 1.19a shows an example of a single-selection list box. The GUI in this figure creates a list box with eight options, labeled "Option 1", "Option 2," and so forth. In addition, the Gm creates a pushbutton to perform selection and a text field to display the selected choice. Both the listbox and the pushbutton generate callbacks. The corresponding callback functions are shown in Figure 19b. If a selection is made in the listbox, then function Listbox1_Callback will be executed.

Figure 1.19(a) Layout of a simple Gm with a listbox, a pushbutton, and a text field. (b) The callback functions for this GUI. Note that selection can occur either by clicking the button or double-clicking on an item in the listbox.
Figure 1.20 The GUI produced by program test_listbox.

This function will check the figure producing the callback (using function gebf) to see if the selecting action was a single-click or a double-click. If it was a single-click, the function does nothing. If it was a double-click, then the function gets the selected value from the listbox, and writes an appropriate string into the text field.

If the pushbutton is selected, then functionButton1_Callbaek will be executed. This function gets the selected value from the listbox, and writes an appropriate string into the text field. The GUI produced by program test_listbox is shown in Figure 1.20.

In an end of chapter exercise, you will be asked to modify this example to allow multiple selections in the list box.

1.4.9 Sliders

Sliders are graphical objects that allow a user to select values from a continuous range between a specified minimum value and a specified maximum value by moving a bar with a mouse. The 'Value' property of the slider is set to a value between min and max depending on the position of the slider.

A slider is created by creating a uicontrol whose style property is 'slider'. A slider may be added to a GUI by using the slider tool in the Layout Editor.

Figure 1.21a shows the layout for a simple GUI containing a slider and a text field. The 'Min' property for this slider is set to zero, and the 'Max' property is set to ten. When a user drags the slider, it automatically calls the function Slider1_Callback, which is shown in Figure 1.21b. This function gets the value of the slider from the 'Value' property and displays the value in the text field. Figure 1.22 shows this Gm with the slider at some intermediate position in its range.
Introduction to Graphical User Interface (GUI) MATLAB 6.5

1.5 Dialog Boxes

A dialog box is a special type of figure that is used to display information or to get input from a user. Dialog boxes are used to display errors, provide warnings, ask questions, or get user input. They are also used to select files or printer properties. Dialog boxes may be modal or non-modal. A modal dialog box does not allow any other window in the application to be accessed until it is dismissed, whereas a non-modal dialog box does not block access to other windows. Modal dialog boxes are typically used for warning and error messages that need urgent attention and cannot be ignored. By default, most dialog boxes are non-modal. MATLAB includes many types of dialog boxes, the most important of which are summarized in Table 1.4. We will examine only a few of the types of dialog boxes here, but you can consult the MATLAB online documentation for the details of the others.
Table 1.4 Selected Dialog Boxes

1.5.1 Error and Warning Dialog Boxes
Error and warning dialog boxes have similar calling parameters and behavior. In fact, the only difference between them is the icon displayed in the dialog box. The most common calling sequence for these dialog boxes is

\[
\text{errordlg(error\_string, box\_title, create\_mode);}
\]

\[
\text{warndlg(warning\_string, box\_title, create\_mode);}
\]

The error\_string or warning\_string is the message to display to the user, and the box\_title is the title of the dialog box. Finally, create\_mode is a string that can be 'modal' or 'non-modal', depending on the type of dialog box you want to create.

For example, the following statement creates a modal error message that cannot be ignored by the user. The dialog box produced by this statement is shown in Figure 1.25.

\[
\text{errordlg('Invalid input values!', 'Error Dialog Box', 'modal');}
\]

1.5.2 Input Dialog Boxes
Input dialog boxes prompt a user to enter one or more values that may be used by a program. Input dialog boxes may be created with one of the following calling sequences.

\[
\text{answer = inputdlg(prompt)}
\]

\[
\text{answer = inputdlg(prompt,title)}
\]

\[
\text{answer = inputdlg(prompt,title,line\_no)}
\]

\[
\text{answer = inputdlg(prompt,title,line\_no,default\_answer)}
\]
Here, prompt is a cell array of strings, with each element of the array corresponding to one value that the user will be asked to enter. The parameter title specifies the title of the dialog box, and line_no specifies the number of lines to be allowed for each answer. Finally, default_answer is a cell array containing the default answers that will be used if the user fails to enter data for a particular item. Note that there must be as many default answers as there are prompts.

Figure 1.26 An input dialog box

When the user clicks the OK button on the dialog box, his or her answers will be returned as a cell array of strings in variable answer. As an example of an input dialog box, suppose that we wanted to allow a user to specify the position of a figure using an input dialog. The code to perform this function would be

```matlab
prompt{1} = 'Starting x position: ';
prompt{2} = 'Starting y position: ';
prompt{3} = 'Width: ';
prompt{4} = 'Height: ';
title = 'Set Figure Position';
default_ans = {'50','50','180','100'};
answer = inputdlg(prompt,title,1,default_ans);
```

The resulting dialog box is shown in Figure 1.26.

1.5.3 The uigetfile and uisetfile Dialog Boxes

The uigetfile and uisetfile dialog boxes are designed to allow a user to interactively pick files to open or save. These dialog boxes return the name and the path of the file but do not actually read or save it. The programmer is responsible for writing additional code for that purpose. The form of these two dialog boxes is

```matlab
[filename, pathname] = uigetfile(filter_spec, title);
[filename, pathname] = uisetfile(filter_spec, title);
```
Parameter `filter_spec` is a string specifying the type of files to display in the dialog box, such as '*.m', '*.mat', and so on. Parameter `title` is a string specifying the title of the dialog box.

![File Open Dialog Box](image)

Figure 1.27 A file open dialog box created by `uigetfile` on a Windows 2000 Professional system.

After the dialog box executes, `filename` contains the name of the selected file and `pathname` contains the path of the file. If the user cancels the dialog box, `filename` is set to zero. The following script file illustrates the use of these dialog boxes. It prompts the user to enter the name of a mat-file, then reads the contents of that file. The dialog box created by this code on a Windows 2000 system is shown in Figure 1.27. (The style of the box varies for different operating systems. It will appear slightly different on a Windows NT 4.0 or UNIX system).

```matlab
[filename, pathname] = uigetfile('*.mat', 'Load MAT File');
if filename ~= 0
    load(fullfile(filename, 'mat'))
end
```

**Good Programming Practice**

Use dialog boxes to provide information or request input in GUI-based programs. If the information is urgent and should not be ignored, make the dialog boxes modal.

### 1.6 Menus

Menus can also be added to MATLAB GUIs. A menu allows a user to select actions without additional components appearing on the GUI display.
### Table 1.5 Important uimenu Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator</td>
<td>A single character specifying the keyboard equivalent for the menu item. The keyboard combination CTRL+key allows a user to activate the menu item from the keyboard.</td>
</tr>
<tr>
<td>Callback</td>
<td>Specifies the name and parameters of the function to be called when the menu item is activated. If the menu item has a submenu, the callback executes before the submenu is displayed. If the menu item does not have submenus, then the callback executes when the mouse button is released.</td>
</tr>
<tr>
<td>Checked</td>
<td>When this property is 'on', a checkmark is placed to the left of the menu item. This property can be used to indicate the status of menu items that toggle between two states. Possible values are 'on' or 'off'.</td>
</tr>
<tr>
<td>Enable</td>
<td>Specifies whether or not this menu item is selectable. If it is not enabled, the menu item will not respond to mouse clicks or accelerator keys. Possible values are 'on' or 'off'.</td>
</tr>
<tr>
<td>Label</td>
<td>Specifies the text to be displayed on the menu. The ampersand character (&amp;) can be used to specify a keyboard mnemonic for this menu item; it will not appear on the label. For example, the string '&amp;File' will create a menu item displaying the text 'File' and responding to the F key.</td>
</tr>
<tr>
<td>Parent</td>
<td>The handle of the parent object for this menu item. The parent object could be a figure or another menu item.</td>
</tr>
<tr>
<td>Position</td>
<td>Specifies the position of a menu item on the menu bar or within a menu. Position 1 is the leftmost menu position for a top-level menu, and the highest position within a submenu.</td>
</tr>
<tr>
<td>Separator</td>
<td>When this property is 'on', a separating line is drawn above this menu item. Possible values are 'on' or 'off'.</td>
</tr>
<tr>
<td>Tag</td>
<td>The 'name' of the menu item, which can be used to locate it.</td>
</tr>
<tr>
<td>Visible</td>
<td>Specifies whether or not this menu item is visible. Possible values are 'on' or 'off'.</td>
</tr>
</tbody>
</table>

Menus are useful for selecting less commonly used options without cluttering up the GUI with a lot of extra buttons. There are two types of menus in MATLAB: standard menus, which are pulled down from the menu bar at the top of a figure, and context menus, which pop up over the figure when a user right-clicks the mouse over a graphical object. We will learn how to create and use both types of menus in this section. Standard menus are created with uimenu objects. Each item in a menu is a separate uimenu object, including items in submenus. These uimenu objects are similar to uicontrol objects, and they have many of the same properties such as Parent, Callback, Enable, and so forth. A list of the more important uimenu properties is given in Table 1.5. Each menu item is attached to a parent object, which is a figure for the top level menus, or another menu item for submenus. All of the uimenu objects connected to the same parent appear on the same menu, and the cascade of items forms a tree.
Figure 1.28(a) A typical menu structure. (b) The relationships among the uimenu items creating the menu.

Figure 1.28 Continued. (c) The Menu Editor structure that generated these menus.

of submenus. Figure 1.28a shows a typical MATLAB menu in operation, while Figure 1.28b shows the relationship among the objects making up the menu. MATLAB menus are created using the Menu Editor. Figure 1.28c shows the Menu Editor with the menu items that generate this menu structure. The additional properties in Table 1.5 that are not shown in the Menu Editor can be set with the Property Editor (propedit). Top-level context menus are created by uicontextmenu objects, and the lower level items within context menus are created by uimenu objects. Context menus are basically the same as standard menus, except that they can be associated with any GUI object (axes, lines, text, figures, etc.). A list of the more important uicontextmenu properties is given in Table 1.6.
1.6.1 Suppressing the Default Menu
Every MATLAB figure comes with a default set of standard menus. If you want to delete these menus from a figure and create your own menus, you must first turn the default menus off. The display of default menus is controlled by the figure's MenuBar property. The possible values of this property are 'figure' and 'none'.

Table 1.6 Important uicontextmenu Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callback</td>
<td>Specifies the name and parameters of the function to be called when the context menu is activated. The callback executes before the context menu is displayed.</td>
</tr>
<tr>
<td>Parent</td>
<td>The handle of the parent object for this context menu.</td>
</tr>
<tr>
<td>Tag</td>
<td>The 'name' of the context menu, which can be used to locate it.</td>
</tr>
<tr>
<td>Visible</td>
<td>Specifies whether or not this context menu is visible. This property is set automatically and should normally not be modified.</td>
</tr>
</tbody>
</table>

If the property is set to 'figure', then the default menus are displayed. If the property is set to 'none', then the default menus are suppressed. You can use the Property Inspector to set the MenuBar property for your GUIs when you create them.

1.6.2 Creating Your Own Menus
Creating your own standard menus for a Gm is basically a three-step process.

1. First, create a new menu structure with the Menu Editor. Use the Menu Editor to define the structure, giving each menu item a Label to display and a unique Tag value. Also, you must manually create the callback string for menu items. This statement is true for MATLAB 6.5. However, MATLAB 6.5 was modified to automatically generate menu callback strings. This manual step is not necessary in MATLAB 6.5 and later. This can be tricky-the best way to create a menu callback is to examine the callback automatically created for a uicontrol, and use it as an example. The proper form of a uimenu callback string is

MyGui('MenultemTag_Callback' ,gcbo, [] ,guidata(gcbo))

where you should substitute the name of your GUI for MyGUI, and the tag value of the menu item for MenultemTag.

2. Second, edit the properties of each menu item using the Property Editor (propedit), if necessary. The most important properties to set are the Callback, Label, and Tag, which can be set from the Menu Editor, so the Property Editor is usually not needed. However, if you need to set any of the other properties listed in Table 1.5, you will need to use the Property Editor. The list of properties that can be set from the Menu Editor has been expanded in MATLAB 6.5, so it is now unnecessary to use the Property Editor. All properties can now be set directly in the Menu Editor in MATLAB 6.5 and later.

3. Third, implement a callback function to perform the actions required by your menu items. You must manually create the callback function for menu items. The process of building menus will be illustrated in an example at the end of this section.
Good Programming Practice

Unlike GUI objects, MATLAB does not automatically create callback strings and stub functions for menu items. You must perform this function manually. Only the Label, Tag, Callback, Checked, and Separator properties of a menu item can be set from the Menu Editor. If you need to set any of the other properties, you will have to use the Property Editor (propedit) on the figure, and select the appropriate menu item to edit.

1.6.3 Accelerator Keys and Keyboard Mnemonics

MATLAB menus support accelerator keys and keyboard mnemonics. Accelerator keys are "CTRL+key" combinations that cause a menu item to be executed without opening the menu first. For example, the accelerator key "0" might be assigned to the File/Open menu item. In that case, the keyboard combination CTRL+o will cause the File/Open callback function to be executed.

A few CRTL+key combinations are reserved for the use of the host operating system. These combinations differ between PC and UNIX systems; consult the MATLAB online documentation to determine which combinations are legal for your type of computer.

Accelerator keys are defined by setting the Accelerator property in a uimenu object.

Keyboard mnemonics are single letters that can be pressed to cause a menu item to execute once the menu is open. The keyboard mnemonic letter for a given menu item is underlined. For top-level menus, the keyboard mnemonic is executed by pressing ALT plus the mnemonic key at the same time. Once the top-level menu is open, simply pressing the mnemonic key will cause a menu item to execute. Figure 1.29 illustrates the use of keyboard mnemonics. The File menu is opened with the keys ALT+f, and once it is opened, the Exit menu item can be executed by simply typing "x".

Keyboard mnemonics are defined by placing the ampersand character (&) before the desired mnemonic letter in the Label property. The ampersand will not be displayed, but the following letter will be underlined, and it will act as a mnemonic key. For example, the Label property of the Exit menu item in Figure 1.29 is 'E&xit'.

1.6.4 Creating Context Menus

Context menus are created in the same fashion as ordinary menus, except that the top-level menu item is a uicontextmenu. The parent of a uicontextmenu

![Image of context menu](image)

Figure 1.29 The menu shown was opened by typing the keys ALT+f, and the Exit option could be executed by simply typing "x".
must be a figure, but the context menu can be associated with and respond to right mouse clicks on any graphical object. Context menus are created using the "Context Menu" selection on the Menu Editor. Once the context menu is created, any number of menu items can be created under it. To associate a context menu with a specific object, you must set the object's UIContextMenu property to the handle of the uicontextmenu. This is normally done using the Property Inspector, but it can be done with the set command as shown below. If Hcm is the handle to a context menu, the following statements will associate the context menu with a line created by a plot command.

```matlab
Hl = plot (x, y) ;
set (Hl,'UIcontextMenu',Hcm);
```

**Reference**