Chapter 8
The Charm of Graphical User Interface

In this chapter we introduce the use of GUIDE, which is the MATLAB Graphical User Interface Development Environment. This tool enables the user to create Graphical User Interfaces (GUI) that can be used to facilitate interaction with your programs.

Introduction

Nowadays we are used to interacting with programs through windows with menus, buttons, drop-down lists, etc. Such interaction tools constitute a graphical user interface (GUI). Graphical user interfaces are simple to use but relatively difficult to program. MATLAB provides a tool, called GUIDE, that helps you in programming a graphical interface.

GUIDE

The GUIDE Layout Editor makes it possible to design GUIs easily by clicking and dragging the GUI components—such as panels, buttons, text fields, sliders, menus, and so on—into the GUI Layout Area. When you create a GUI, GUIDE generates two files: a FIG-file and an application M-file. The FIG-file contains a description of the GUI appearance, whereas the M-file contains the code that controls the behavior of the GUI. FIG- and M-files need to be stored in the same folder. GUIs are governed by callback functions, which are routines within the M-file that are executed when a specific event occurs in any of the elements of the GUI (i.e., button, drop-down list, etc.). GUI's elements are called UiControls.

M. Borgo et al., *MATLAB for Psychologists*,
DOI 10.1007/978-1-4614-2197-9_8, © Springer Science+Business Media, LLC 2012

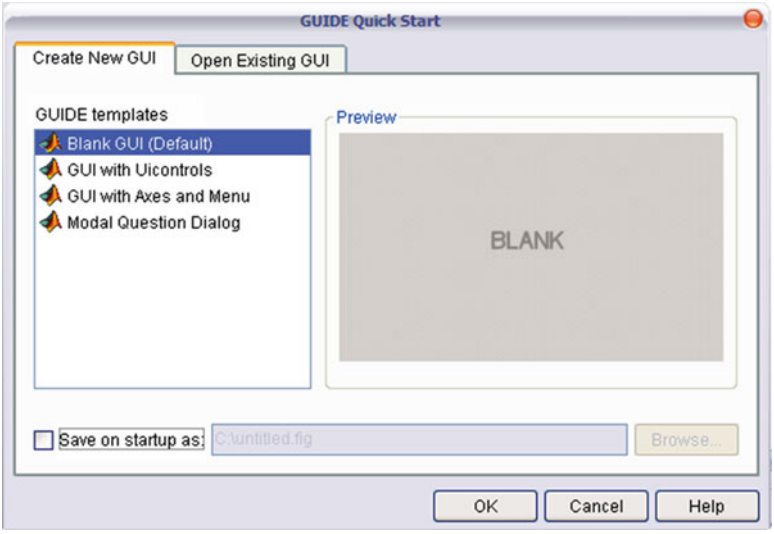


Fig. 8.1 GUIDE Quick Start dialog form

Starting GUIDE

Start GUIDE by typing `guide` at the MATLAB prompt or by selecting File -> New -> GUI from the menu bar. MATLAB displays the GUIDE Quick Start dialog box, as shown in Fig. 8.1. From the Quick Start dialog, you can choose from among four different GUIDE templates, that is, prebuilt GUIs that can be later modified at your convenience.

Choose the Blank GUI (Default) and ignore the 'Save on startup as' option at the bottom left corner of the box. A GUI is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools. Figure 8.2 shows the Layout Editor for the blank GUI template. (If there are no names in your GUI component palette, don't panic, read the preferences for GUIDE section, later in this chapter.)

We can design the GUI by dragging components, such as panels, pushbuttons, pop-up menus, or axes, from the component palette on the left side of the Layout Editor into the layout area. Before going into the details of these components, we will go through the key menus, providing a short description of the main menu items.

The GUI Toolbar

GUIs come with a Toolbar with a number of shortcuts so that you can reach directly some of the options that are otherwise available from the Menu Bar. Figure 8.3 shows the icons in the GUI toolbar with their meanings.

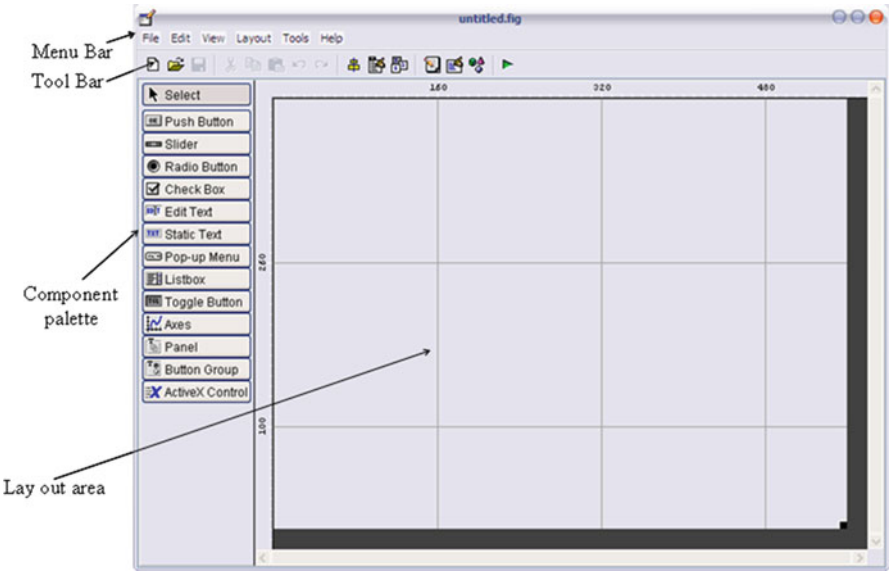


Fig. 8.2 Layout Editor for the blank GUI template

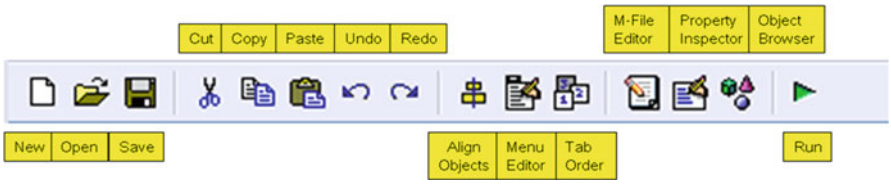


Fig. 8.3 The GUI toolbar

Adding UiControls to the GUI

Once we have created the GUI, it's time to insert the UiControls in the layout area to implement the GUI's behavior. Table 8.1 shows what the GUI UiControls do:

In addition to the Callback functions associated to each component, GUIDE generates three further functions within the application M-file. The first has the name of the GUI and we usually do not act on it. The other two functions are `nameOfFile_OpeningFcn()` and `nameOfFile_OutputFcn();` we need to operate on these two functions to save users' inputs. Refer to the Saving users' inputs section to see how this can be done.

Table 8.1 The component palette

Component palette	UiControl	Description
	Select	Restores the mouse pointer to selection mode
	Push button	This component creates a button that is used to take action in the GUI. Most of the GUIs you'll create will contain pushbuttons such as Cancel, OK, or Save
	Slider	This component creates a UiControl that can be manipulated with the mouse
	Radio button	A radio button together with an accompanying label is created by this component. Use this component if you need to choose one option within a pool of options. When two or more radio buttons are grouped within a button area, the selection of one of them causes the other to be automatically deselected.
	Check box	With this component you create a check box together with an accompanying label. You can select or clear the check box to turn on or off the behavior that has been programmed for this component
	Edit text	This component creates a text box into which the user can type text. You can also use a text box to display text to the user, or to provide text for the user to copy and paste elsewhere. A text box can contain either one line or multiple lines. In this latter case, a vertical scroll bar will also appear
	Static text	With this component you create a label, text used to identify a part of the GUI or to show information to the user
	Pop-up menu	With this component you create a box from which the user can choose from a list of options. They are much like the menus on the menu bar; except that they can be placed anywhere in the GUI
	List box	With this component you create a list box, which is a component that lists a number of values. The user can pick one value from the list, but can't enter a new value
	Toggle button	With this component you create a button that is used for taking action in the GUI. Unlike the pushbutton, the toggle button remains selected after being pressed, signaling whether the option is active
	Axes	With this component you create a box in which a plot can be inserted
	Panel	With this component you create a frame, an area of the GUI surrounded by a thin line together with an accompanying label. Use a panel to group related elements in the GUI
	Button group	With this component you create a frame as with the Panel component, but button groups manage *exclusive* selection for radio buttons and toggle buttons. That is, when a button group contains a number of radio buttons or toggle buttons, by selecting one of them all the others within its area are automatically deselected
	ActiveX	This control opens a select ActiveX control box in which you can select an ActiveX control to insert into your GUI (for Windows users only)

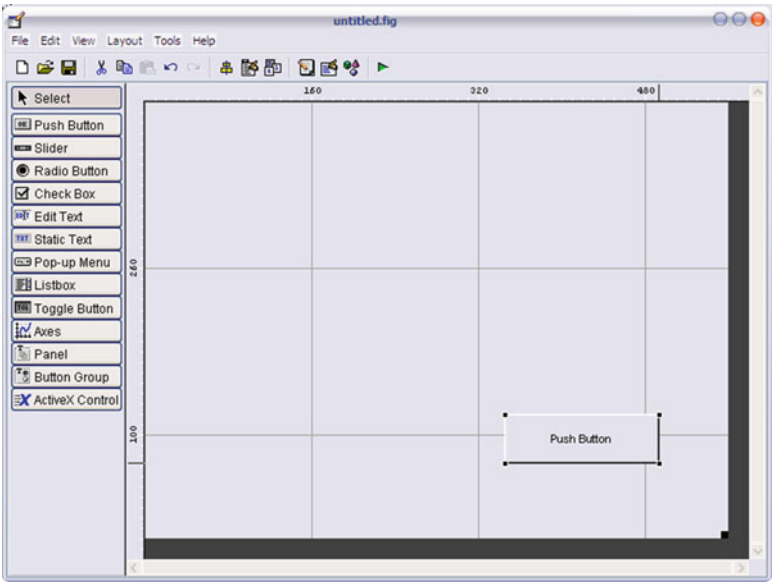


Fig. 8.4 Placing a Push Button

Closing the GUI

As a first example, let us implement a Cancel pushbutton; that is a UiControl that just closes the GUI. Run guide from the command line and chose Blank GUI (Default) from the GUIDE Quick Start box. The GUI is now opened in the Layout Editor. Select the pushbutton icon in the component palette and place it in the GUI layout area as shown in the Fig. 8.4.

By default, GUIDE names pushbuttons "`Push Button`". By double clicking on the pushbutton, the Properties Inspector box appears. Now change both the String and the Tag property to `Cancel` (Fig. 8.5).

This step needs to be commented. Do not confuse String and Tag properties. The String property is the string that will appear on the pushbutton. The Tag property, instead, is the actual name of the UiControl within the MATLAB code. In other words, when you need to refer to a UiControl, you have to call it with its Tag, not with its String.

To keep things simple, we can give to both the Tag and the String property the same name, but this is not always possible. For example, in the String property you can use any character you like (including periods, commas, and symbols). In contrast, in the Tag property you are restricted to letters of the alphabet.

There is another important thing to notice. After saving the GUI to the hard drive, the GUIDE automatically renames the Callback property from the default "`%automatic`" to `filename('Cancel_Callback',gcbo, [] ,guidata(gcbo))`.

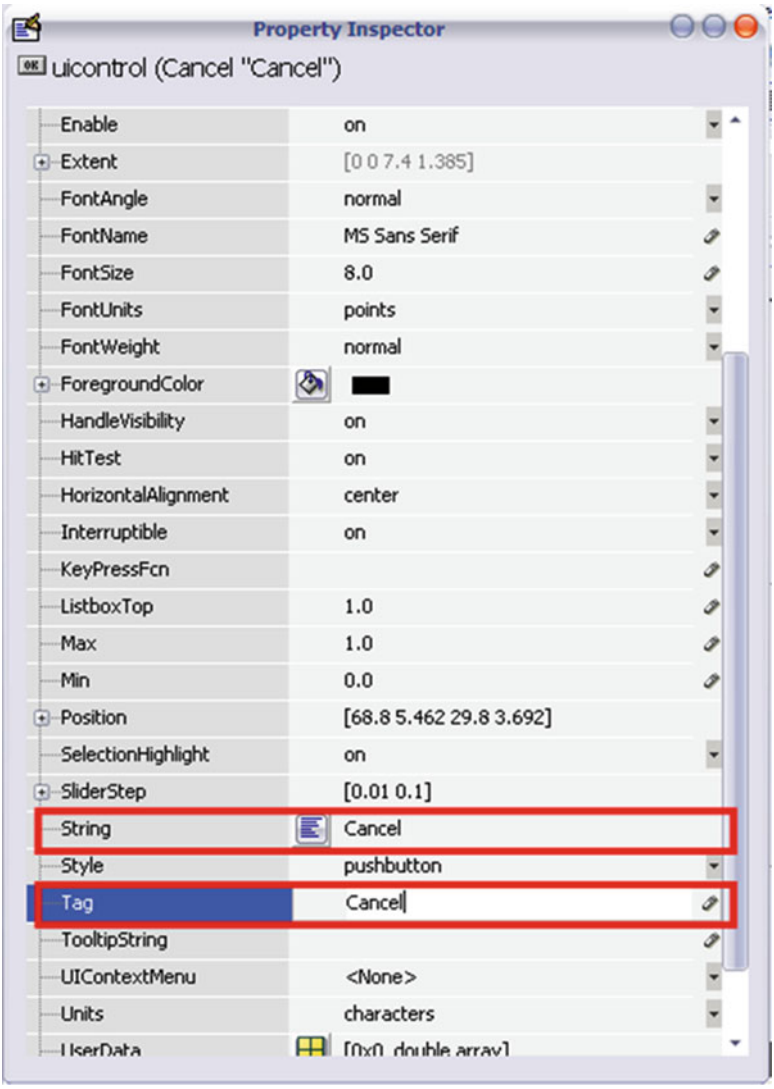


Fig. 8.5 Change both the String and Tag property in "Cancel"

Figure 8.6 shows what happens to the Property Inspector box after you save the GUI to the hard drive, naming it ClosingGUI.

This makes the finding of the Callback function for the Cancel pushbutton within the application M-file easier. The following lines of code were generated by GUIDE after saving the GUI.

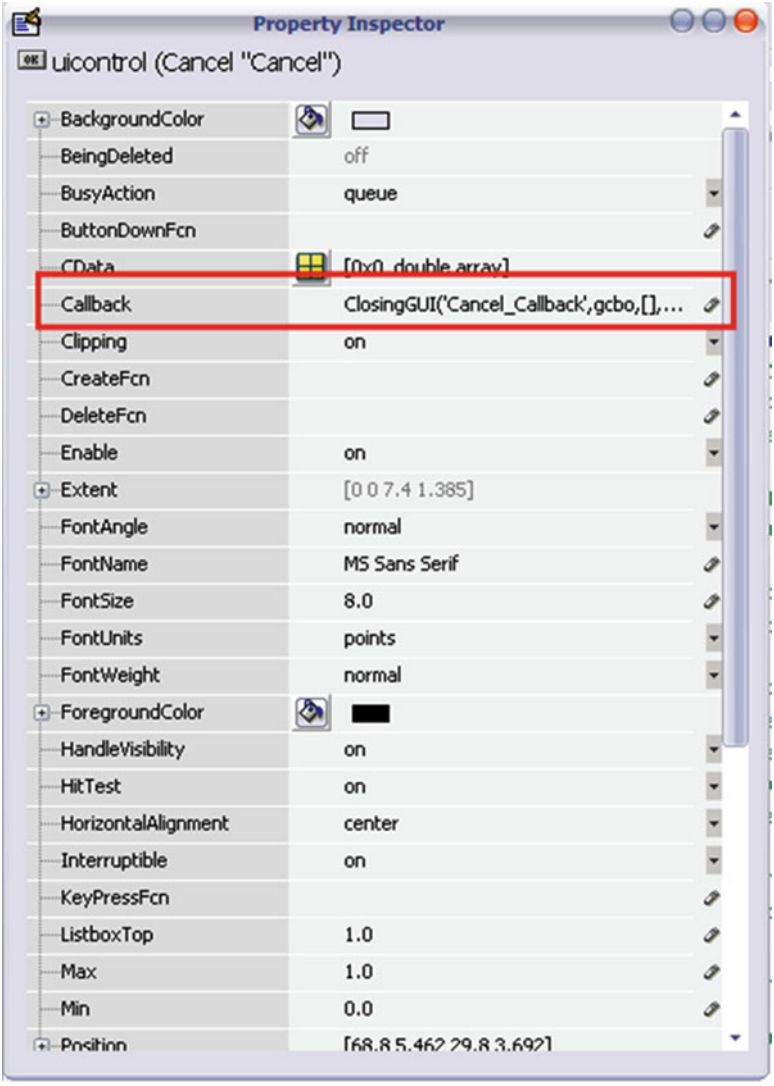


Fig. 8.6 After saving the GUI with a specific name and after naming the Tag property of the pushbutton, GUIDE automatically updates the callback function name accordingly

```
% --- Executes on button press in Cancel.
function Cancel_Callback(hObject, eventdata, handles)
% hObject handle to Cancel (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

Select the Cancel pushbutton and choose, from the View menu items, Callbacks-> Callback. You are directly pointed into the Cancel_Callback function in the application M-file.

After the automatically generated comments, add `delete(gcf);` as shown:

```
% --- Executes on button press in Cancel.
function Cancel_Callback(hObject, eventdata, handles)
% hObject handle to Cancel (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
delete (gcf);
```

The `gcf` command returns the handle to the current figure (see Chap. 5), i.e., the GUI image, and the delete function deletes the argument that is passed to it.

Now we are ready to run the GUI.

Select Tools -> run or press Ctrl+T, or click on the  icon in the Tool Bar. MATLAB will ask you to save the changes. Say yes, and your first GUI is running. When you click on the Cancel button, the GUI closes!

This is not very exciting, but you should have now an idea on how GUIs work. In the next sections we will see something more relevant, such as controlling the appearance of UiControls from other UiControls, inserting figures and graphs in the GUI, and saving participants' input.

Controlling UiControls from Other UiControls

In this section we see how to sum two numbers that have been typed by the user using a pushbutton. This is a very instructive example because it shows how the various components of the GUI communicate with one another.

The Sum-Two-Numbers Example

Create a GUI and place the Cancel button, as we have done before. (It is always a good habit to insert a Cancel button into your GUIs. It might happen that you decide not to run the program after all.) Proceed as in the previous section:

1. Run GUIDE from the command line;
2. Choose Blank GUI (Default) from the GUIDE Quick Start box;
3. Select the pushbutton icon from the component palette;
4. Change the name of both the Tag and the String property to 'Cancel';
5. Select the pushbutton and add `delete(gcf);` in the Cancel_Callback function in the application M-file.
6. Save and name the GUI SumTwoNumbers.

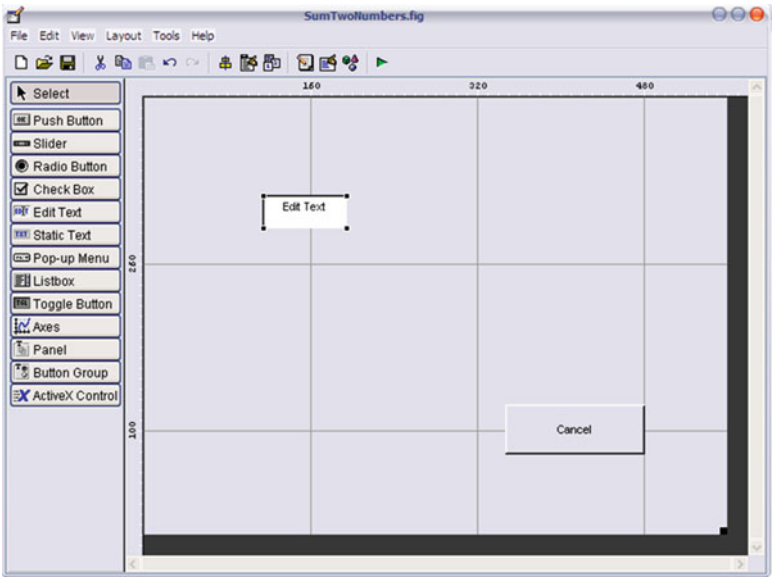


Fig. 8.7 Add an Edit Text component into the GUI Layout Editor

From the component palette select the Edit Text component and drag it into the GUI Layout Editor as shown in Fig. 8.7.

By default, "Edit Text" appears in the text box. We do not need that text. To delete the text, look at the properties of the text box within the Property Inspector box and cancel the "Edit text" value from the String property. You now have a blank Edit Text component in the GUI Layout Editor.

As you can see from the Property Inspector window, the default Tag for this component is "edit1". Because there will be more than one edit text component in this GUI, change the name to "FirstNumber" (Fig. 8.8).

From the Component palette select a Static Text component and drag it just below the FirstNumber component. Replace its String property from "static text" to "First Number". Since we are not going to work with this component, we can leave its Tag property as "text1". Select both the FirstNumber and the text1 components and use the Align Objects tool to horizontally align them (Fig. 8.9).

Repeat the procedure twice to add another Edit Text and Static Text component pair. From the Property Inspector box, delete the String property of the first Edit Text component and replace its Tag from "edit2" to "SecondNumber". Replace the Tag of the other Edit Text component from "edit3" to "Total". Select the first Static Text component so that the Property Inspector box shows its properties and replace its String property from "static text" to "SecondNumber". Replace the String property of the other Static Text component from "static text" to "Total". Use the Align Objects tool to vertically align and distribute these components. The GUI now should look similar to the one shown in Fig. 8.10.

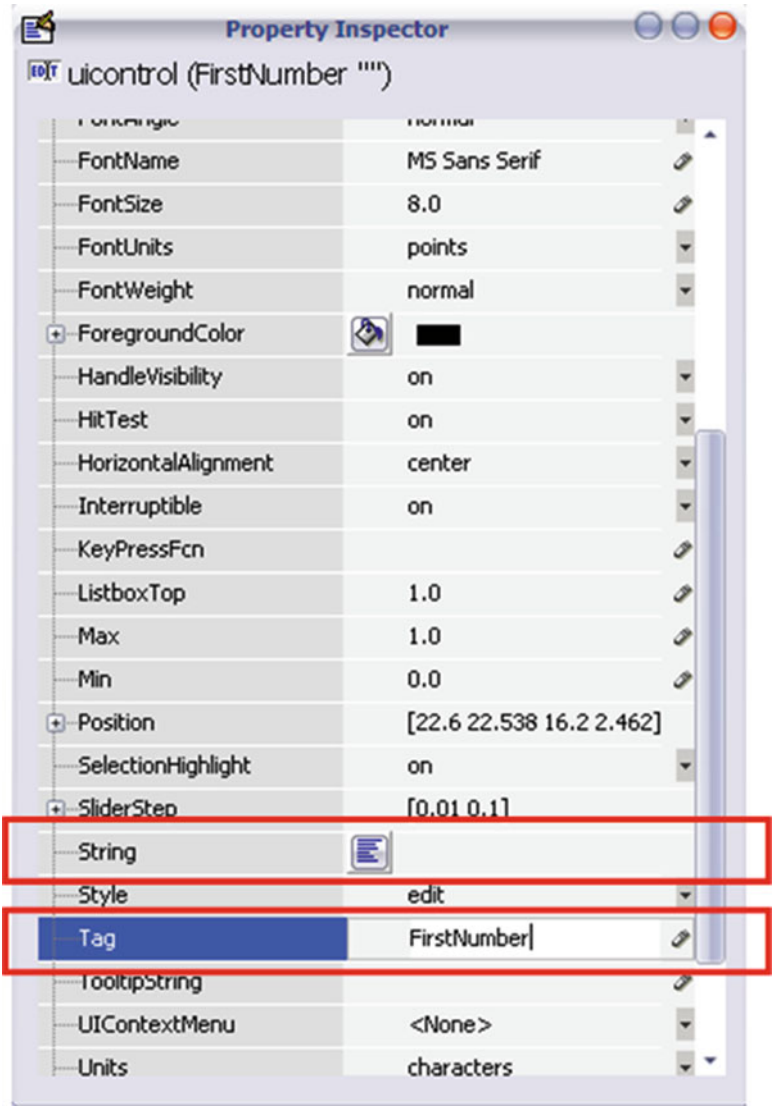


Fig. 8.8 Delete the default "Edit Text" in the String property and change the Tag property from "edit1" to "FirstNumber"

The last component we need is another pushbutton, and this will be biggest part of the job. Follow the same procedure that you followed to create the Cancel push-button. Name both its Tag and its String properties "Sum".

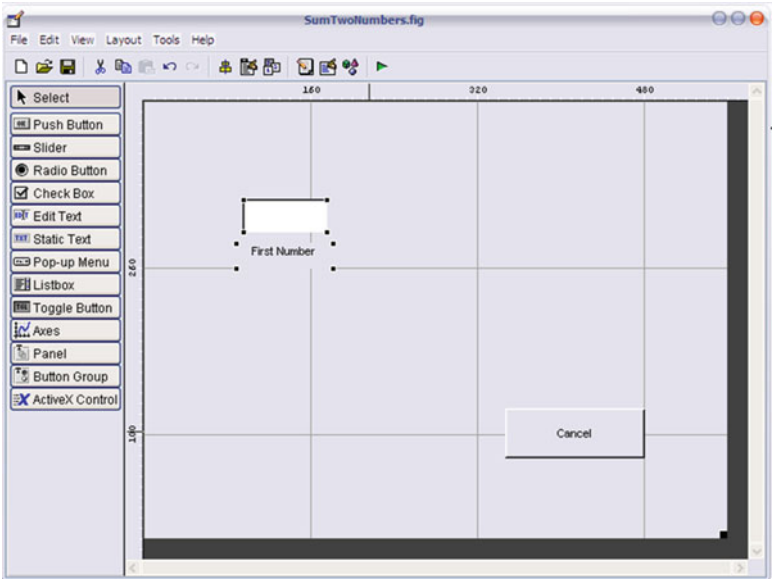


Fig. 8.9 Add a static text component and change its String property from edit1 to First Number

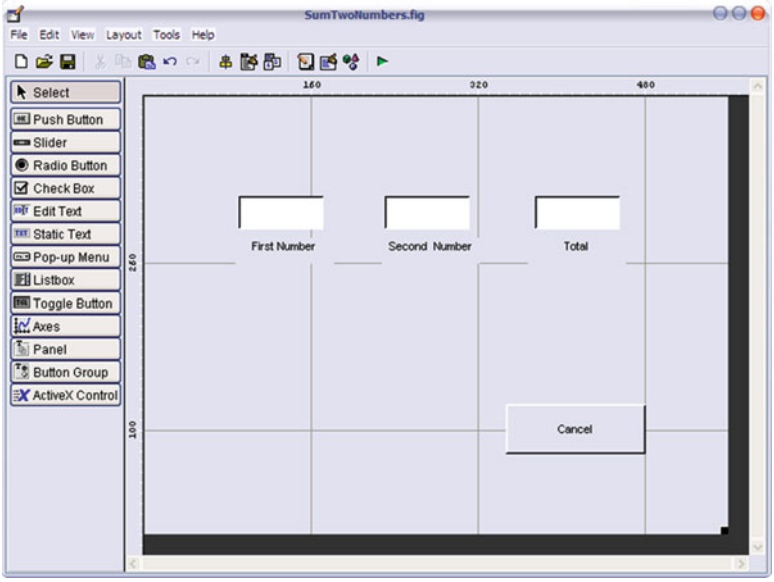


Fig. 8.10 Add other two Edit Text and Static Text components

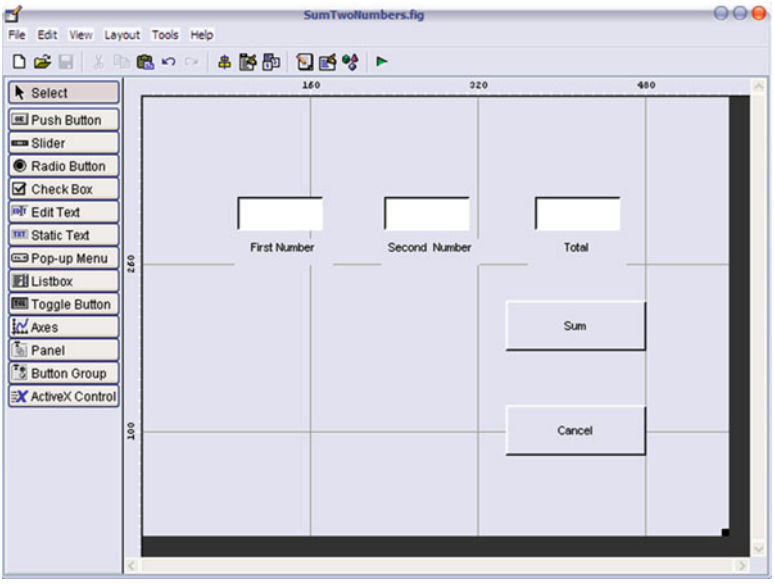


Fig. 8.11 The GUI Layout Editor with all the components you need for this task

The design of the GUI is now complete and should look similar to the one shown in Fig. 8.11.

You are now ready to program the GUI's behavior. Select the Sum button and from the View menu item choose View Callbacks -> Callback. You are directed to the application M-file, into the Move_Callback function.

After the comments, type the following line of code as shown:

```
function Sum_Callback(hObject, eventdata, handles)
% hObject handle to Sum (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

FirstNumber = str2double(get(handles.FirstNumber,'String') );
SecondNumber = str2double(get(handles.SecondNumber,'String') );
sum=FirstNumber+SecondNumber;
set(handles.Total,'String',sum);
```

`FirstNumber` and `SecondNumber` are variables that have been created to store user inputs. User input is character type; since we need numbers instead of characters, we change it by means of the `str2double()` function. The `get()` function, together with its counterpart `set()` function, is a very important one because it is used to `get` the values of the property that we are referring to. In this case, we want to `get` the `'String'` values that have been typed in the `FirstNumber` and `SecondNumber` UiControls. Note that to refer to these components, we used the handles structure (which is passed to the callback function as argument), followed

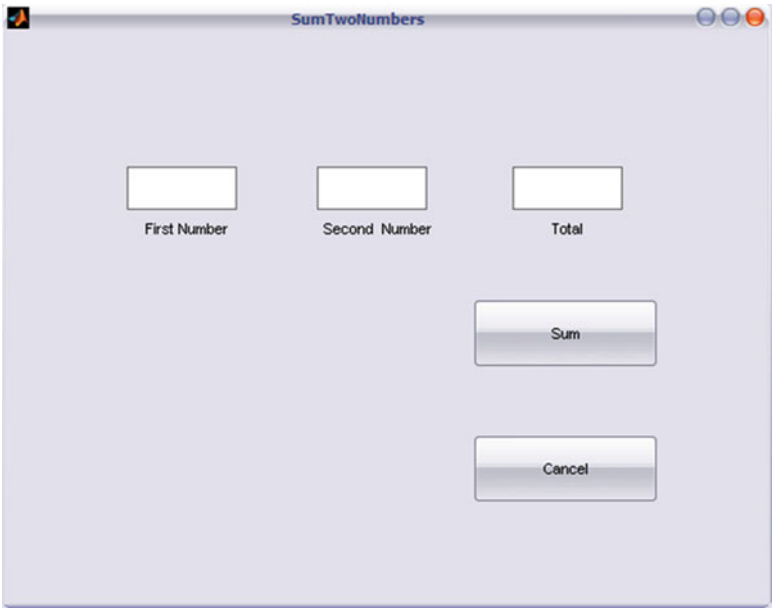


Fig. 8.12 The SumTwoNumbers GUI running. After typing two numbers, press the Sum button to display their sum in the Total edit box

by a dot and then the Tag. To refer to the property we need, we add a comma followed by the property name within quotation marks.

In the third line, we saved the sum of the numbers in the `sum` variable. Finally, in the last line we used the `set()` function to *set* the String property of the Total UiControl to the sum of the two numbers.

While running, this GUI should appear as in Fig. 8.12.

In this example we have seen how to *get* and *set* the properties of one component from the Callback function of another component. It is also possible to *set* and *get* the property of one component from *within* its own Callback function. In this case, instead of using the `handles` structure, we use `hObject`, which is the first argument that is passed to Callback functions.

We can improve the SumTwoNumber GUI by changing the background color of the Sum pushbutton when it is pressed. In this way, the user knows whether it has been pressed at least once.

Within the same `Sum_Callback(hObject, eventdata, handles)` add the following line of code:

```
set(hObject,'BackgroundColor','red')
```

Run the GUI again and press the Sum button. As you can see, besides summing the numbers, now the Sum button is turned to red (Fig. 8.13).

We could have done the same job using the `handles` argument of the same `Sum_Callback` function. However, when using the `hObject` argument, we do not need to specify the UiControl Tag.

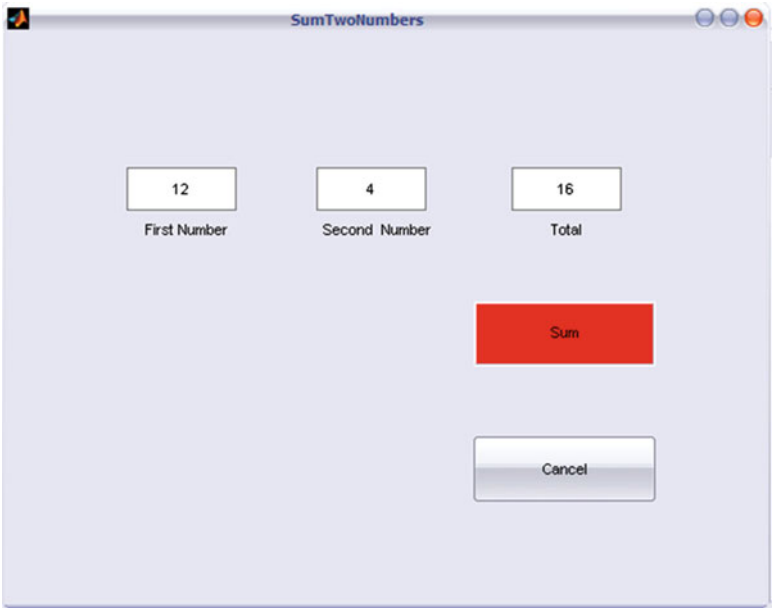


Fig. 8.13 The update SumTwoNumbers GUI. When the Sum button is pressed, it turns red

Displaying Graphs and Figures in the GUI

Among the components, it is the `axes` component that serves to display graphs and figures. Graphs are quite easy to understand. The following example shows how to plot a line in the axes component by pressing a button.

- Create a GUI and place the Cancel button.
- Save and name the GUI DisplayGraph.

From the component palette select the components and drag them into the GUI Layout Editor as shown in Fig. 8.14.

From the Property Inspector box, name the Tag for the upper Edit Text "To" and the bottom one "From". Name the first pushbutton "Display". Add the following lines of code in the `Display_Callback` function:

```
myfrom = str2double(get(handles.From,'String')) ;
myto = str2double(get(handles.To,'String'));
plot(myfrom:myto);
```

When the Display graph pushbutton is pressed, a line is plotted in the `axes component`.

This example is not very appealing. However, you get the picture now; you can plot any type of graph you like following these instructions. Displaying figures is another interesting feature of GUI. To explain how this works, in the following example we show how to display pictures representing visual illusion, into our GUI.

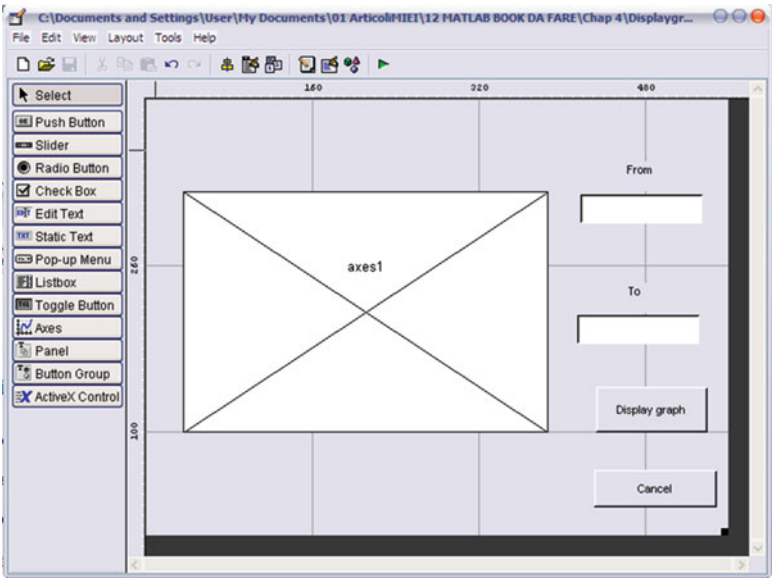


Fig. 8.14 GUI to display a graph

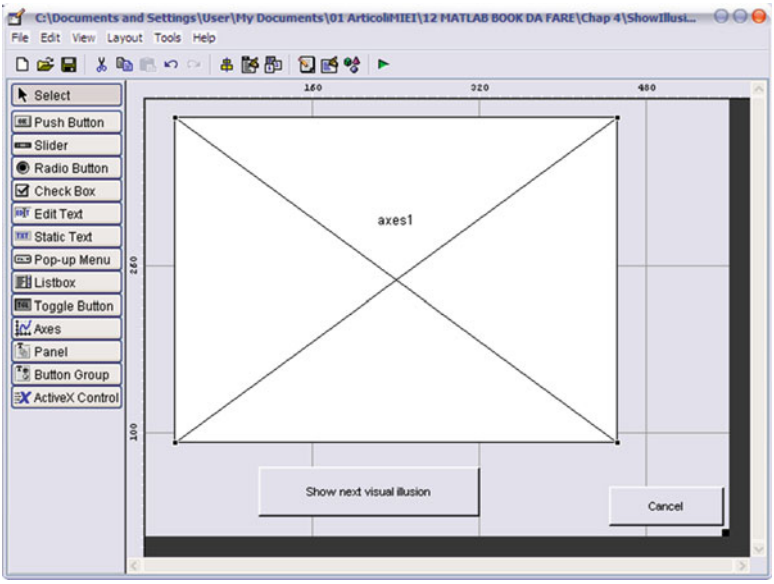


Fig. 8.15 GUI to show visual illusion saved in the current directory

Place your favorite visual illusion in the current directory in JPEG format (any other format will do). In MATLAB create a GUI and place a large axes component, the usual Cancel button, and a pushbutton whose tag is ShowNextIllusion (see Fig. 8.15).

In the ShowIllusions_OpeningFcn function add the following line of code that implements the variable count.

```
Global count;
```

In the `ShowNextIllusion_Callback` function add the following lines of code to display the JPEG figure that you have in your current directory.

```
global count
count=count+1;
illusions=dir(fullfile(cd,'*.jpeg'));
if count>size(illusions,1)
    msgbox('Sorry, no more illusions to show')
else
    illusionToshow=illusions(count).name;
    img=imread(illusionToshow);
    image(img)
end
```

Saving User Input

When running experiments, psychologists need to save participants' input. There are different ways of doing this. The simplest is to include a `save()` function within a pushbutton Callback function. You then retrieve participants' data using the `load()` (see Chap. 2) function from outside the GUI. However, you may want to save and retrieve participants' input while running the experiment, so you want to send participants' input to different M-files or to the MATLAB console. In these cases, there are a few steps to follow, and various functions of the application M-file have to be manipulated.

To show how this works, we consider the SumTwoNumbers GUI again, but now we want to store the numbers that have been typed by the user. We could do this within the `Sum_Calback` function, but to make it clearer, it is perhaps better to create another pushbutton and name both its Tag and its String "Save" as shown in Fig. 8.16.

The GUI design is now complete. Now we need to program its behavior.

There are three steps to follow:

Step 1

In the `OpeningFcn` function uncomment the `%uiwait(handles.figure1);` line as shown.

```
function SumTwoNumbers_OpeningFcn (hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle editto figure
% eventdata reserved - editto be defined in a future version of MATLAB
```

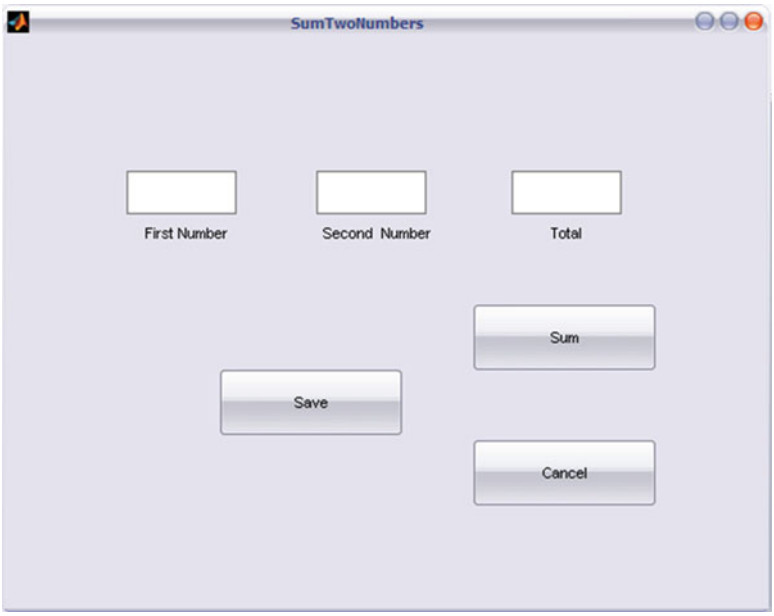


Fig. 8.16 Create a Save pushbutton into the SumTwoNumbers GUI Layout editor

```
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments editto SumTwoNumbers (see VARARGIN)

% Choose default command line output for SumTwoNumbers handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes SumTwoNumbers wait for user response (see UIRESUME)
uiwait(handles.figure1);
```

The `uwait()` function puts the GUI in standby mode. To understand what this means can be helpful in clarifying what happens when this command is commented. Without this command, MATLAB jumps directly into the `SumTwoNumbers_OutputFcn` function without waiting for the user's input. You do not want this to happen, because you want the OutputFcn function to be called only *after* the users have typed their numbers, so that this function "knows" what has been typed.

Step 2

In the SumTwoNumbers_OutputFcn function, replace `varargout{1}=handles.output;` with `varargout{1}=handles;` and add "`delete(gcf);`", as follows:

```
function varargout = SumTwoNumbers_OutputFcn(hObject,eventdata,handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle editto figure
% eventdata reserved - editto be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)

% Get default command line output editfrom handles structure
varargout{1} = handles;
delete(gcf);
```

This code allows for saving different variables in a structure. By substituting `handles.output` with `handles`, we are instructing the OutputFcn function to output our own structure (we will come back to this point shortly). The `delete (gcf);` command is the same as the one within the `Cancel_Callback` function. By adding the command here as well, we are closing the GUI when the Save pushbutton is pressed.

Now the program is ready to save anything we want and to close the GUI after the save operation. The last step is to tell GUIDE what we want to save. In the `Save_Callback` function add the following lines of code:

Step 3

```
function Save_Callback(hObject, eventdata, handles)
% hObject handle to Save (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
handles.Number1 = get(handles.FirstNumber,'String');
handles.Number2 = get(handles.SecondNumber,'String');
guidata(hObject, handles);
uiresume(handles.figure1);
```

The `handles.Number1=get(handles.FirstNumber,'String');` and `handles.Number2=get(handles.SecondNumber,'String');` commands save the two numbers into the new `Number1` and `Number2` fields in the `handles` structure.

The `guidata(hObject, handles);` command updates the handles structure with the new field.

Finally, `uiresume()` reactivates the GUI so that the `OutputFcn` function can be executed. Note that if we close the GUI here, by means of `the delete()` function instead of using `uiresume()`, we do not save anything, because the `OutputFcn` function needs to get the updated handles structure to output it.

Let us now see how the program works. From the MATLAB command prompt type:

```
mystruct=SumTwoNumbers;
```

Now the program is running; type any number in the two edit box components and then press the Save pushbutton. When you do this, the GUI closes, and a structure named `mystruct` has been created in the MATLAB workspace. To retrieve the numbers that have been typed, just use the mystruct.Number1 and mystruct. Number2 variables.

Adding Your Own Functions

To conclude this chapter, it has to be noted that within the M-file you can insert any additional functions you want, and of course, you can pass to these functions both the `hObject` and the `handles` structures as arguments. Refer to Exercise 2 for an example.

Summary

- GUIDE generates two files that save and launch the GUI: a FIG-file and an application M-file.
- GUIDE owns a number of dedicated preferences that can be set from the Preferences dialog box in the File menu.
- GUIs are shaped by dragging components from the component palette into the layout area.
- A component's properties are set from the Properties Inspector box.
- GUIs benefit from Object Orienting Programming (OOP) technology, and their `UiControls` are governed by means of callback functions.
- Callback functions receive three arguments: `hObject`, `eventdata`, `handles`.
- The `get()` and `set()` functions are used to get and set, respectively, a component's properties.
- The `uiwait()` and `uiresume()` functions are used to stop and reactivate the GUI.
- The `Filename_OutputFnc()` function returns to MATLAB any input the user provides.

Exercises

Exercise 1

Using Radio Buttons grouped within a Button Group component, alter the visible property of two Static Text components so that they appear either to the left or to the right of the GUI.

Solution

Place the components as shown in Fig. 8.17.

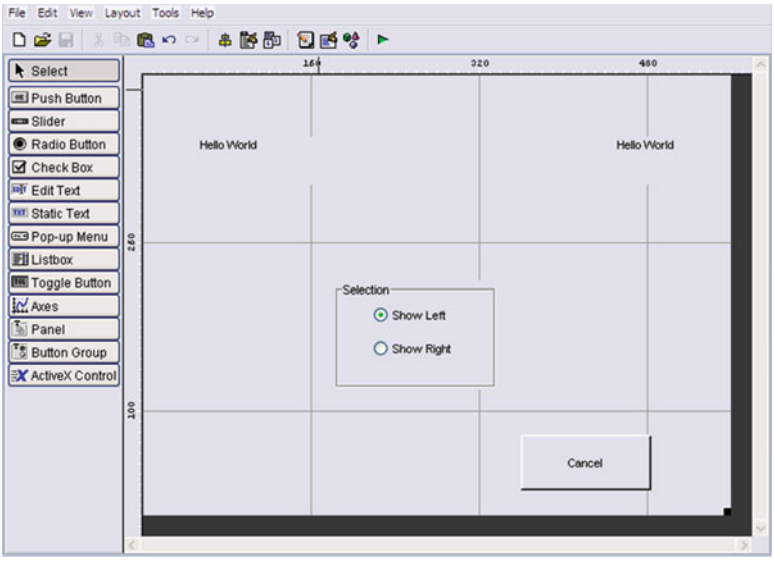


Fig. 8.17 Example of a GUI as required by Exercise 1

Assign the Tags to the Static text components "Left" and "Right". Assign the Visible property of both of them to "off". Name the Tags for the two radio buttons ShowLeft and ShowRight. Place the following lines of code within uipanel1_SelectionChangeFcn function:

```
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
switch get(hObject,'Tag')
    case 'ShowLeft'
       set(handles.Left,'Visible', 'on')
       set(handles.Right,'Visible', 'off')
    case 'ShowRight'
       set(handles.Left,'Visible', 'off')
       set(handles.Right,'Visible', 'on')
end
```

Since the two Radio buttons are within the Button Group component, when the GUI is executing, selecting one Radio button automatically deselects the other.

Exercise 2

Suppose you are doing a memory experiment and that a participant has been presented with a sample of figures. Program a GUI to present a collection of figures (some of which where presented in the original sample and others that weren't) within the Axes component and add two pushbuttons. The participant has to press

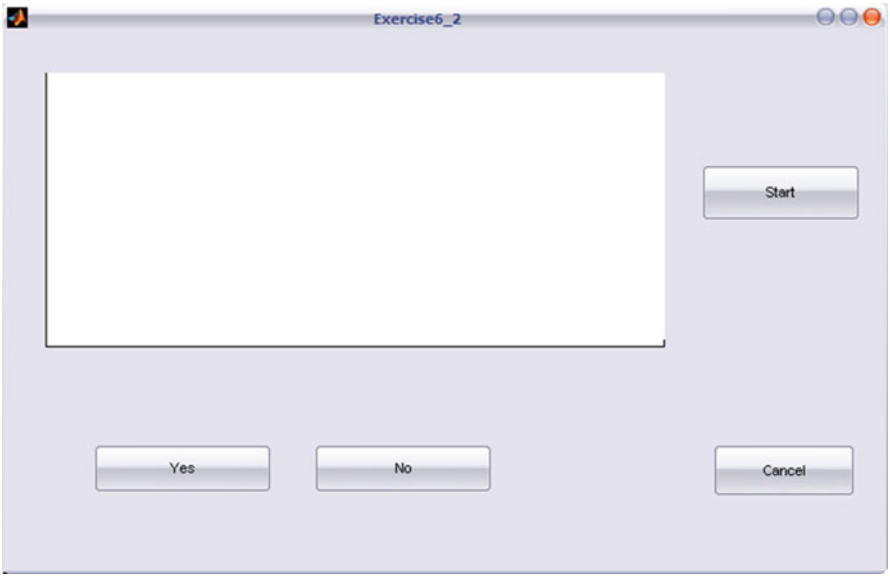


Fig. 8.18 GUI example as required by the exercise

one or the other pushbutton depending on whether the figure that is shown was present in the original sample or not. The participants' responses are to be collected and coded as right or wrong.

Solution

There are many different ways to solve this exercise. The one that we present here makes use of the handle structure created by GUIDE (but other solutions are possible) and creates a new function (named "`savedata`") in the application M-file that has been created by GUIDE, which takes as arguments both the handles and the answer that has been given by the participant.

Place the figures you want to present in the current directory and name them in such way that is easy for you to recall whether they were present in the original sample (for example, "A1.jpeg" for the first "Absent" figure and "P1.jpeg" for the first Present figure). Create a GUI like the following one (Fig. 8.18).

Here is the code for this GUI. Call this file from the MATLAB prompt and save the structure in the `mystruct` variable. `mystruct.response` and `mystruct.figure` are cell arrays containing participants' answers and figure names, respectively. We will then analyze these data by comparing participants' answers with the initials (either A or P) of the figure names.

Listing 8.1

```
function varargout = Exercise6_2(varargin)

% EXERCISE6_2 M-file for Exercise6_2.fig
% EXERCISE6_2, by itself, creates a new EXERCISE6_2
% or raises the existing singleton*.
%
% H = EXERCISE6_2 returns the handle to a new EXERCISE6_2 or the handle to
%the existing singleton*.
%
% EXERCISE6_2('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in EXERCISE6_2.M with the given input arguments.

% EXERCISE6_2('Property','Value',...) creates a new EXERCISE6_2 or
% raises the existing singleton*.  Starting from the left, property value
% pairs are applied to the GUI before Exercise6_2_OpeningFunction gets
% called.  An unrecognized property name or invalid value makes property
% application stop.
% All inputs are passed to Exercise6_2_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu.
% Choose "GUI allows only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
%
% Edit the above text to modify the response to help Exercise6_2
%
% Last Modified by GUIDE v2.5 18-Feb-2010 19:59:06
%
% Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct( 'gui_Name',       mfilename, ...
        'gui_Singleton',  gui_Singleton, ...
        'gui_OpeningFcn', @Exercise6_2_OpeningFcn, ...
        'gui_OutputFcn',  @Exercise6_2_OutputFcn, ...
        'gui_LayoutFcn',  [] , ...
        'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end
    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT


% --- Executes just before Exercise6_2 is made visible.
function Exercise6_2_OpeningFcn(hObject, eventdata, handles, varargin)

    % This function has no output args, see OutputFcn.
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to Exercise6_2 (see VARARGIN)
    % Choose default command line output for Exercise6_2
    handles.output = hObject;
    handles.count=0;
    figures=dir(fullfile(cd,'*.jpeg'));
    handles.response=cell(size(figures,1),1);
```

```
61      handles.myfigure=cell(size(figures,1),1);
62      % Update handles structure
63      guidata(hObject, handles);
64
65      % UIWAIT makes Exercise6_2 wait for user response (see UIRESUME)
66      uiwait(handles.figure1);
67
68
69      % --- Outputs from this function are returned to the command line.
70  function varargout = Exercise6_2_OutputFcn(hObject, eventdata, handles)
71
72      % varargout  cell array for returning output args (see VARARGOUT);
73      % hObject    handle to figure
74      % eventdata  reserved - to be defined in a future version of MATLAB
75      % handles    structure with handles and user data (see GUIDATA)
76
77      % Get default command line output from handles structure
78      varargout{1} = handles;
79      delete(gcf);
80
81
82      % --- Executes on button press in Yes.
83  function Yes_Callback(hObject, eventdata, handles)
84
85      % hObject    handle to Yes (see GCBO)
86      % eventdata  reserved - to be defined in a future version of MATLAB
87      % handles    structure with handles and user data (see GUIDATA)
88
89      savedata(hObject, 'Yes', handles)
90
91
92      % --- Executes on button press in No.
93  function No_Callback(hObject, eventdata, handles)
94
95      % hObject    handle to No (see GCBO)
96      % eventdata  reserved - to be defined in a future version of MATLAB
97      % handles    structure with handles and user data (see GUIDATA)
98
99      savedata(hObject, 'No', handles)
100
101     %--------------------------
102 function savedata(hObject,answer,handles)
103
104     handles.count=handles.count+1;
105     guidata(hObject, handles);
106     figures=dir(fullfile(cd,'*.jpeg'));
107
108     if handles.count>size(figures,1)
109         msgbox('End experiment')
110         guidata(hObject, handles);
111         uiresume(handles.figure1);
112     else
113         handles.myfigure(handles.count)={figures(handles.count).name};
114         handles.response(handles.count)={answer};
115         img=imread(handles.myfigure{handles.count});
116         image(img)
117         axis off
118     end
119     guidata(hObject, handles);
120
121     % --- Executes on button press in Start.
122 function Start_Callback(hObject, eventdata, handles)
123     % hObject    handle to Start (see GCBO)
124     % eventdata  reserved - to be defined in a future version of MATLAB
```

(continued)

Listing 8.1 (continued)

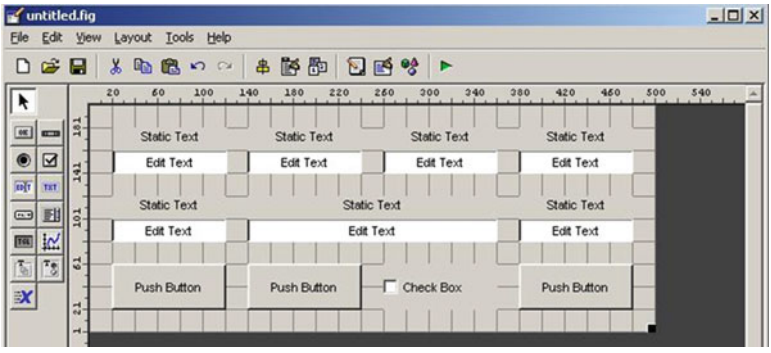
```
    % handles    structure with handles and user data (see GUIDATA)
    figures=dir(fullfile(cd,'*.jpeg'));
    figureToshow=figures(1).name;
    img=imread(figureToshow);
    image(img)
    axis off

   % --- Executes on button press in Cancel.
function Cancel_Callback(hObject, eventdata, handles)
   % hObject    handle to Cancel (see GCBO)
   % eventdata  reserved - to be defined in a future version of MATLAB
   % handles    structure with handles and user data (see GUIDATA)
   delete(gcf)
```

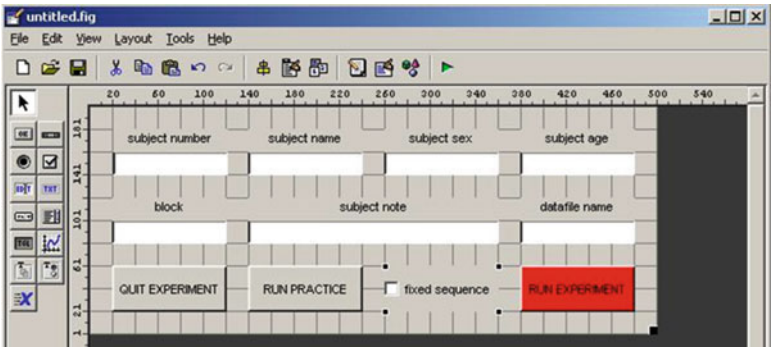
A Brick for an Experiment

We just read how we can build a graphical interface. We may want to build a graphical interface to drive our experiment. For example, we may want to have a graphical interface that allows us to collect the subject's identifying information (e.g., the name, sex, age, as well as the number we assign to the subject) as well as an optional text note about the subject (e.g., is the subject naive or expert? right-handed or left-handed?). Additionally, we may want to have a button to "START" the experiment. We may want also to have the possibility to run instead of the complete experiment, only an abbreviated version of it with a few trials (one for each experimental condition) so that we can run a few practice trials if we need to. We may want/need to add further items to our graphical interface. For example, it could be useful to set within it the name of the file where we are going to save the data or the block number if the experiment we run is subdivided into more than one block. And we could insert other parameters as well, for example if we want stimuli to be presented in fixed or random order. Of course, the number of things you can control through a graphical interface has no limit.

The graphical interface we build is rather standard, so that we will set the following parameters: subject's number; subject's name; subject's sex; subject's age; subject's note; the block number; the datafile's name; the possibility to run the experiment in a fixed or random order; one "start experiment" button; one "start practice" button; one "cancel" button. Now launch the guide command and begin to draw a graphical interface that looks more or less like the following one:



Now edit the text properties of the various objects you have created as follows, and in addition, change the bottom-right pushbutton background color to red:



Now change the tag properties of your "Edit Text" slots and give them (more) meaningful names such as nsub, subname, subsex, subage, nblock, subnote, and filename. Now change the tag properties of your pushbuttons into QuitExp, RunPractice, and RunExp. Finally, change the tag property of your check box into isfixed.

The first thing we do is to activate the "QUIT EXPERIMENT" button. We saw how to do this above. Just insert the command line delete(gcf) into the QuitExp_ Callback function. Now activate the GUI for the user's input by uncommenting the uiwait(handles.figure1) command (see the Save user input section). In this way, the GUI now waits for user input. We now have to replace "varargout{1}=handles. output;" with "varargout{1}=handles;" so that we can get all the data we input each time we run the experiment within a structure called "handles". Immediately after this command, we write again the delete(gcf) command. In this way, we close the GUI if we press the "RUN EXPERIMENT" button. Do the same after the RunPractice_Callback function. Now we move to the end of the RunExp_Callback function. Here, after the comments, we have to add the following lines of code:

```
handles.nsub = str2num(get(handles.nsub,'String'));
handles.subname = get(handles.subname,'String');
handles.subsex = get(handles.subsex,'String');
handles.subage = str2num(get(handles.subage,'String'));
handles.nblock = str2num(get(handles.nblock,'String'));
handles.subnote = get(handles.subnote,'String');
handles.filename = get(handles.filename,'String');
handles.isfixed = get(handles.isfixed,'Value') == get(handles.
isfixed,'Max');
guidata(hObject, handles);
uiresume(handles.figure1);
delete(gcf)
```

Now save as "RunExp" your GUI. Note that we converted nsub and subage from strings to numbers, whereas subname and subsex are strings. Go to the command line and type the following command (remember to comment temporarily delete(gcf) first):

```
>> UserInputs = RunExp
```

Now input a set of hypothetical subjects' details and press the button "RUN EXPERIMENT". MATLAB should echo the contents of a structure. Within this content there are also the subject's number, the subject's name, the subject's sex, and so on.

Now that we have collected all the user's inputs, we have to pass them to the main program that runs the experiment. This program was introduced in Chap. 4 (Listing 4.19). To do this, we need to transform that script into a function that receives as input the data structure created through the graphical interface. So, add the following line at the top of that listing:

```
function SekulerExp(InputDataStruct)
```

Now let us return to the graphical interface. There, we need to add a further command line that will be executed when we press the "RUN EXPERIMENT" button at the end of the RunExp_Callback function (i.e., just after the lines of code we wrote previously):

```
SekulerExp(handles);
```

In practice, when we press the "RUN EXPERIMENT" button, we will launch the SekulerExp function, which receives a structure as input data. Within this structure will be all the subject's details, as well as the datafile name, the isfixed value, and so on. The last thing we do is to make the "run practice" button effective. Add the following lines after the RunPractice_Callback function:

```
handles.nsub = 0;
handles.isfixed = get(handles.isfixed,'Value') == get(handles.isfixed,'Max');
SekulerExp(handles);
```

You can see that here, rather than getting the data that the user inputs in the GUI, we directly set the nsub variable to zero. In the SekulerExp function, by convention, when the subject number is set to zero, the program runs a practice experiment, an experiment with one repetition for each of the stimuli.

The graphical interface is ready. Within the SekulerExp function, we can get the input data by adding the following lines at the beginning of the code.

```
% EXPERIMENT'S SETTINGS
% get input data from the structure passed through the interface
nsub = InputDataStruct.nsub;
subname = InputDataStruct.subname;
subsex = InputDataStruct.subsex;
subage = InputDataStruct.subage;
```

```
nblock = InputDataStruct.nblock;
subnote = InputDataStruct.subnote;
isfixed = InputDataStruct.isfixed;
filename = InputDataStruct.filename;
```

Let's now take a look at the renovated SekulerExp function:

Listing 8.2

```
function SekulerExp(InputDataStruct)

% M-script to realize a experiment based on the crossmodal perception
% The experiment first performed by Sekuler, Sekuler and Lau (1997)
% Author: Borgo, Soranzo, Grassi 2009

% EXPERIMENT'S SETTINGS
% get input data from the structure passed through the interface
nsub = InputDataStruct.nsub;
subname = InputDataStruct.subname;
subsex = InputDataStruct.subsex;
subage = InputDataStruct.subage;
nblock = InputDataStruct.nblock;
subnote = InputDataStruct.subnote;
isfixed = InputDataStruct.isfixed;
filename = InputDataStruct.filename;
% set the experiment details
conditions = [1, 1; 1, 2; 2, 1; 2, 2];
repetitions = 20;
if nsub == 0
   repetitions = 1;
end
EventTable = GenerateEventTable(conditions, repetitions, isfixed);
TotalNumberOfTrials = length(EventTable(:, 1));
for trial = 1:TotalNumberOfTrials

   % STIMULI (SELECTION)
   VideoStimulusToPlay = EventTable(trial, 2);
   SoundStimulusToPlay = EventTable(trial, 3);

   % STIMULI (CREATION)
   % STIMULI (PRESENTATION)
   % COLLECT SUBJECT'S ANSWER
end

% STORE RESULTS
```

Appendix

Referring to Fig. 8.2, we give a short description of the remaining menu items that weren't previously defined. You can find the same information in the MATLAB help; we have given them here for quick reference.

The File Menu

As you might guess, the File menu provides commands for handling files. Here's a sketch of the items in the File menu.

New (Ctrl+N) Displays the GUIDE Quick Start dialog box again; that is, you can create more than one GUI at a time.

Open (Ctrl+O) Displays the usual open dialog box to open FIG-files you have already created.

Close (Ctrl+W) Closes the GUI.

Save (Ctrl+S) and Save as… save the current GUI to disk. This item will appear as Save and the name of the GUI, so you can clearly tell from the menu which open GUI it is being saved. It should be noted that when a GUI is saved, two files are automatically saved: The FIG-file and the M-file, both with the same name. However, once it is saved, to run our GUI we need only to prompt its name without any extensions (or you can double click in the FIG-file icon).

Export… This item is very helpful if you want to save one M-file only instead of both an M-file and a FIG-file. When you select this option, MATLAB first saves the current GUI to disk, that is, both the M- and FIG-files. Then it saves another M-file whose code creates the GUI from scratch. The default name for this M-file is the same as that of the saved M-file plus the "_export" suffix. You can change it according to your needs.

Preferences… displays the same dialog box that is displayed from the Preferences item in the MATLAB main windows (see the Preferences for GUIDE section in this chapter).

Print (Ctrl+P) Displays the Print dialog box for printing the GUI figure.

The Edit Menu

The Edit menu provides commands for working in the GUI. Most of these commands are standard to mainstream applications.

Undo (Ctrl+Z) undoes the previous action. GUIDE supports multiple undo operations; simply continue undoing to undo further actions.

Redo Redoes the last undone action. Again, GUIDE supports multiple redo operations, up to the number of undo operations that have been done.

Cut (Ctrl+X) Deletes the selected UiControl from the GUI and copies it into the Clipboard, allowing for pasting it in a different position of the same GUI or into another GUI.

Copy (Ctrl+C) Copies the selected UiControl to the Clipboard, allowing for pasting a copy of it in a different position of the same GUI or into another GUI.

Paste (Ctrl+V) Pastes the UiControl from the Clipboard into the current GUI.

Clear Deletes the selected UiControl.

Select All (Ctrl+A) Selects all the UiControls in the current GUI.

Duplicate (Ctrl+D) Duplicates the selected UiControl.

The View Menu

The View menu provides the means for displaying and moving the various windows of the GUI. Some of these windows are context sensitive, that is, the displayed window differs according to the selected UiControl. Here are the View menu items:

Property inspector Displays the Property inspector window. Property inspector is an interactive tool for exploring and modifying a UiControl's property values.

Object browser displays a hierarchical list of the UiControls in the GUI. You can select any UiControl from here.

M-File editor displays the application M-file connected with the GUI. If you haven't already saved the GUI, the Save as dialog box will first appear.

View Callbacks is similar to the previous one because it displays the application M-file connected with the GUI, but here you have the opportunity to jump directly into the callback function that you need. For example, if you need to change a callback function of a given pushbutton, you first select it and then, from the View Callback menu item, you jump directly into its callback function prototype.

The Layout Menu

The layout menu works on the selected UiControls by snapping them to the grid or by moving them backward and forward. We use this last feature when there are UiControls overlapping each other.

Snap to grid ties UiControls to the grid square borders when moved. (The Tools -> Grid and Rulers… menu item allows for displaying the grid and changing its size.)

Bring to Front (Ctrl+F) moves the selected UiControl(s) in front of the others.

Send to Back (Ctrl+B) moves the selected UiControl(s) to the back of the others.

Bring Forward moves the selected UiControl(s) forward by one level, that is, not in front of all UiControl, as Bring to Front does, but only in front of the one overlapping

it. Hence, if you have three overlying uincontrols and you want to bring the last one in the second level, this is the item you need to use.

Send Backward moves the selected UiControl(s) back by one level, that is, behind the UiControl directly behind it, but not behind all UiControls, as Send to Back does.

Tools Menu

The Tools menu provides commands for running the GUI, to align UiControls, to display and regulate the grid and the ruler, to create Menus into the GUI, to display the Tab Order editor box, and to set the GUI options.

Run (Ctrl+T) starts running the current GUI.

Align Objects… displays the Align Objects box. This tool allows for aligning and distributing the UiControls within the GUI both vertically and horizontally. Facility with this tool will save considerable time. In practice, when you want to align or distribute two or more UiControls, you first to select them, you then click onto the self-explanatory icon in the Align Objects box, and finally you press the Apply button. The Align option aligns the selected UiControls to the same reference line, while the Distribute option spaces the selected UiControls uniformly with respect to each other. By default, the UiControls are spaced within the bounding box, but you can also space them to a specified value in pixels by selecting the Set spacing option and specifying the pixel value.

Grid and Rulers… displays the Grid and Rulers box. This tool allows for displaying the rulers and the grid in the GUI background. You can also regulate the Grid Size by selecting the desired pixel value for each square from the Grid Size pop-up menu. In this box there is also a duplicate of the snap-to-grid option that we have already discussed in the Layout Menu item. Of course, neither the ruler nor the grid will appear in run mode.

Menu Editor… displays the Menu Editor box. With this option you add Menus and menu items, in addition to context menus into your GUIs. Menus, menu items, and context menus work similarly to UiControls; that is, they perform the action defined in their Callback functions. As for the UiControls, for menus as well, prototype Callback functions are automatically created in the M-file by GUIDE.

Help

The help menu provides two items for help and information.

Using the Layout editor displays a starting guide on “Creating graphical user interfaces.” It is a sort of index from which you can select many different subguides.

Creating GUIs displays a list of sections on how to create graphical user interfaces (GUIs) using GUIDE.

Preferences for GUIDE

There are a number of preferences that you can set for GUIDE. These preferences can be found in three different locations within the Preferences dialog box, which can be invoked from the File menu.

Confirmation preferences:
GUIDE can display a confirmation dialog box when "saving changes" is needed for GUIDE to proceed. Basically, before running (activating) the GUI and before exporting it, any change that has been done has to be saved. If you think that you may not want to keep these changes, then from the MATLAB file menu, select General -> Confirmation Dialogs to access the GUIDE confirmation preferences and tick on "`prompt to save on activate`" and/or "`prompt to save on export`" as shown in Fig. 8.19.

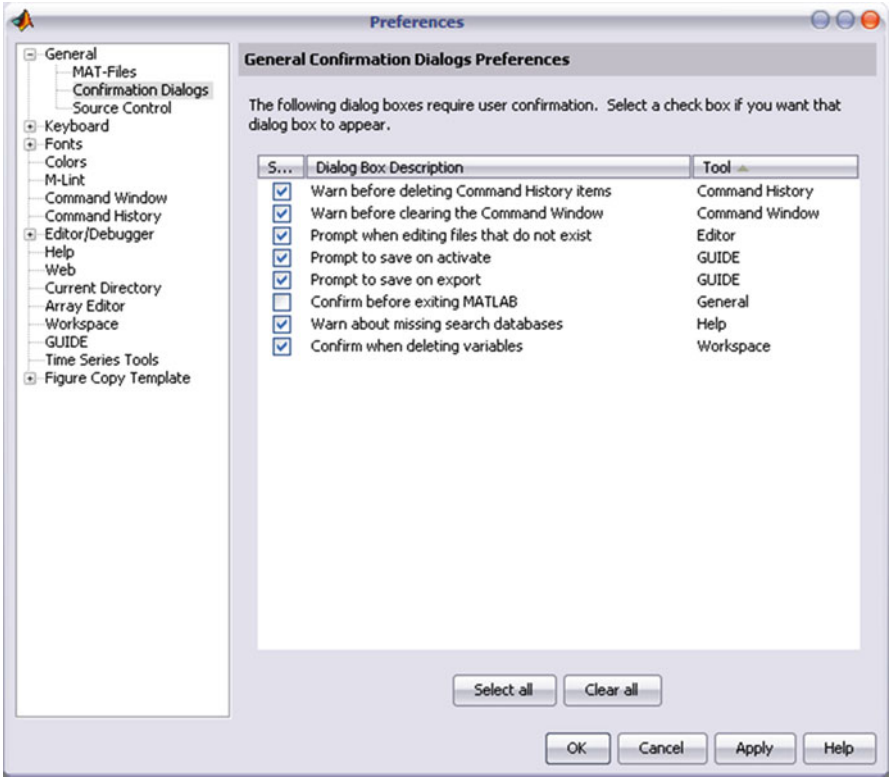


Fig. 8.19 Confirmation preferences

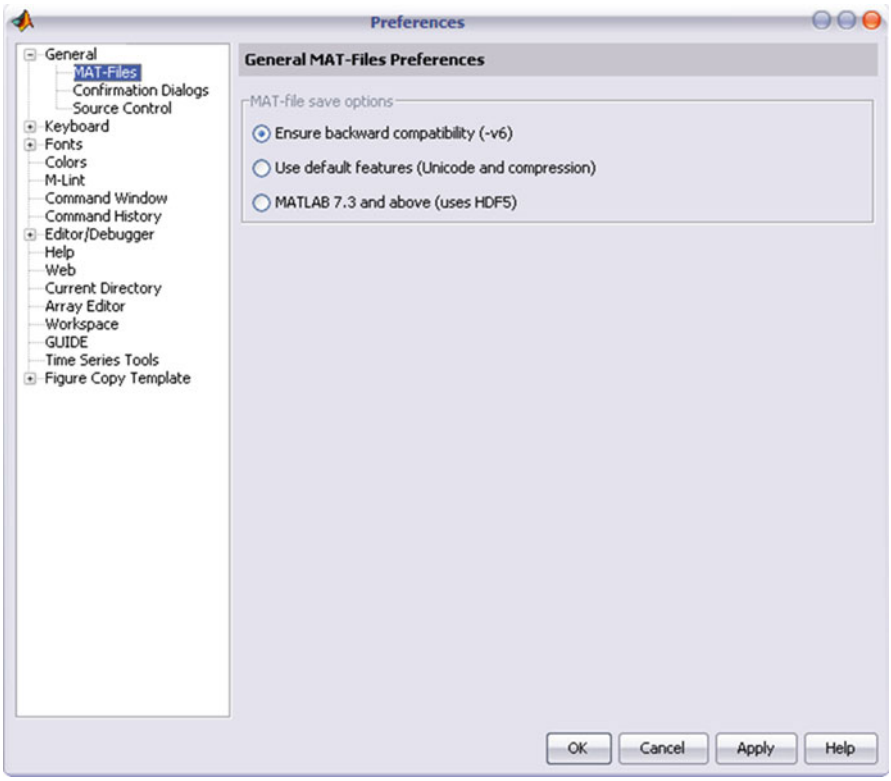


Fig. 8.20 Ensure backward compatibility (-v6)

Backward Compatibility

If you created a GUI with MATLAB 7.0 or an earlier version, and you need to run it also with older MATLAB versions, then this is the preference that you want.

From the MATLAB File menu, select Preferences and then click on `Ensure backward compatibility (-v6)` in the Preferences dialog box under General > MAT-Files (Fig. 8.20).

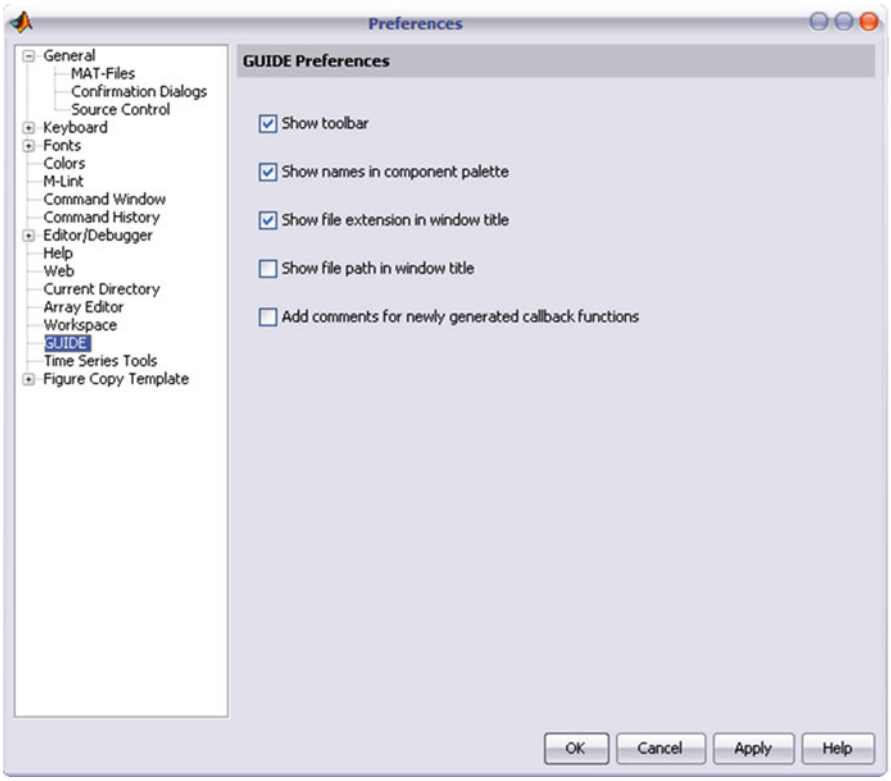


Fig. 8.21 GUIDE preferences

Other Preferences

Finally, five additional preferences for the Layout Editor interface and for M-file comments can be set from the Preferences dialog box, by selecting GUIDE in the left-hand panel. These preferences are self-explanatory. Unless you are already familiar with GUIs, it might be useful to click on "Add comments" for newly generated callback functions (Fig. 8.21).

Suggested Readings

Marchand P, Holland OT (2002) Graphics and GUIs with MATLAB. Boca Raton, FL: Chapman & Hall/CRC

Smith ST (2006) MATLAB: Advanced GUI development. Indianapolis, IN: Dog Ear Pub