2

Using MATLAB to develop standalone graphical user interface (GUI) software packages for educational purposes

A. B. M. Nasiruzzaman
Department of Electrical & Electronic Engineering,
Rajshahi University of Engineering & Technology
Bangladesh

1. Introduction

In the institutes where laboratory facilities are not that much available, and industries are located in remote areas, Personal Computer (PC) can be used to facilitate science and engineering education. Programming and simulation tools can be used widely for preparing such a PC based setting for students. But to develop a software, toolbox or standalone applications one had to rely on C++, Visual basic, or Java. For a computer science or information technology student it is easy to program in these environments but for other science and engineering students this pose a problem since they are not familiar with these programs and require excellent programming expertise. MATLAB (MathWorks, 2009), flagship software in scientific computing, is extensively used all over the world. Particular factors that support the selection of MATLAB are:

• A flexible software structure of MATLAB comprising libraries, models, and programs enables one to integrate different model components in one package conveniently.

• Fast development with MATLAB using powerful calculation and visualization means of the package enables one to expand the software quickly and efficiently without developing any extra programming tools.

• A wide selection of TOOLBOXes, comprehensive collections of predefined functions for solving application-specific problems, is already available with MATLAB and is likely to grow even faster in the future.

The use of MATLAB (short for MATrix LABoratory) is increasing day by day (McMohan, 2007; Littlefield & Hanselman, 2004). Science and engineering students use this software broadly for educational purposes (Chapman, 2007). Graphical User Interface (GUI) is an environment available with renowned software that gives the option to the user developing software packages for personal and problem specific uses. It is a way of arranging information on a computer screen that is easy to understand and use because it uses icons, menus and a mouse rather than only text and programs written in high level language which is often not much handy for others except for programmers. The rapidly developing software MATLAB for technical computation is giving two releases per annum with extended capabilities which enhances user performance and boosts customer satisfaction. Collaterally, its size is increasing. With every release MATLAB takes a new look with new features and changes. Each version is accompanied by major bug fixes, enhanced help menus, removal of undocumented deprecated functions, and development of alternative functions. In case of GUI this change is more rapid, functions are being obsolete and new efficient functions are generated. Due to this reason a GUI developed in one version may not be used in other version (Marchand, 2002; Smith, 2006). So other ways must be taken into deliberation to solve this setback. This chapter presents a guide to develop standalone software tools and/or packages using the enormous competence of MATLAB GUI which can be used in educational and training institutes for learning purposes of freshman or sophomore students.

But another problem arises simultaneously, the huge size and memory requirements of these new releases of MATLAB prevents its uses on the PCs having low memory. Sometimes the programs developed by the recent versions of MATLAB cannot be used by the previous ones due to lack of version compatibility option. So, although instructors can develop interactive tutorial packages for students using the recent versions of MATLAB which they can afford easily, the developed software cannot be used in the laboratories having older facilities. Students also cannot take the advantage of using the new software since he does not have the financial capacity of purchasing newer versions or upgrading MATLAB and high performance PC. To solve this hindrance MATLAB standalone project development tools can be used.

This chapter describes the development of an interactive computer based GUI for MATLAB which can be used in any Pentium III graded PC. It has been prepared for anyone who has little or no exposure to MATLAB. Readers are guided through new concepts to build easy-to-use GUI, acting as a 'wrapper' for experimental simulation codes written by the educator. Even though the chapter is written based on the recent release of MATLAB 2009a, this can be used as a guide for other versions starting from MATLAB 6. It is designed to relieve the coder from most of the programming burden, and to provide with a friendly, consistent approach to the development of standalone MATLAB programs.

2. Getting Started

MATLAB GUI can be built in two ways.

(a) Using GUIDE (GUI Development Environment)
(b) Coding from MATLAB editor

The first approach of building GUI is straightforward and will be discussed in this chapter. Once completed several examples of building GUI, anyone can learn how to code from MATLAB editor.

To initiate GUIDE let's write *guide* in the MATLAB command window and press ***Enter*** key. This will open the GUIDE Quick Start window as shown in Fig. 1, where there are two tabs (Create NEW GUI and Open Existing GUI). Under the Create New GUI tab four GUI templates are available. Selecting the Blank GUI (Default) template and pressing Ok at the bottom of the window opens the design window as shown in Fig. 2.

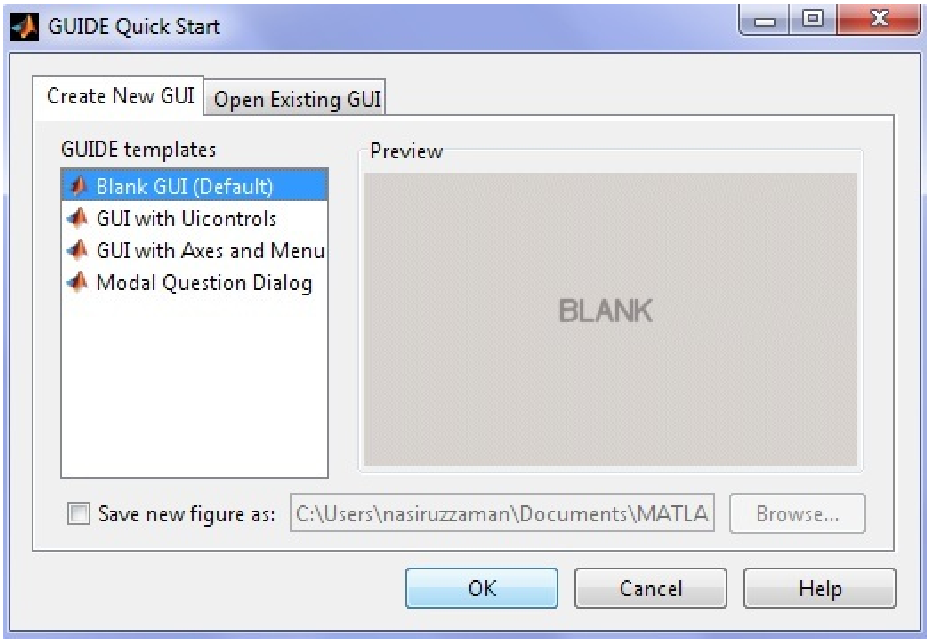


Fig. 1. GUIDE quick start window

The GUI is not yet saved, so at the top of the window it is shown *untitled.fig*. Once the work is saved the title of the GUI will be reflected here. *.fig is the extension of GUI figure files. Generally, a GUI requires two files the figure (*.fig) files where various components are aligned and the code (*.m) files where the coding is done. There is also provision to run the GUI using the single *.m file. At the top of the window Menu and Shortcuts can be found. To the right there are some GUI controls which are very important to learn for building GUI. The blank portion is used for the design purpose of the GUI.

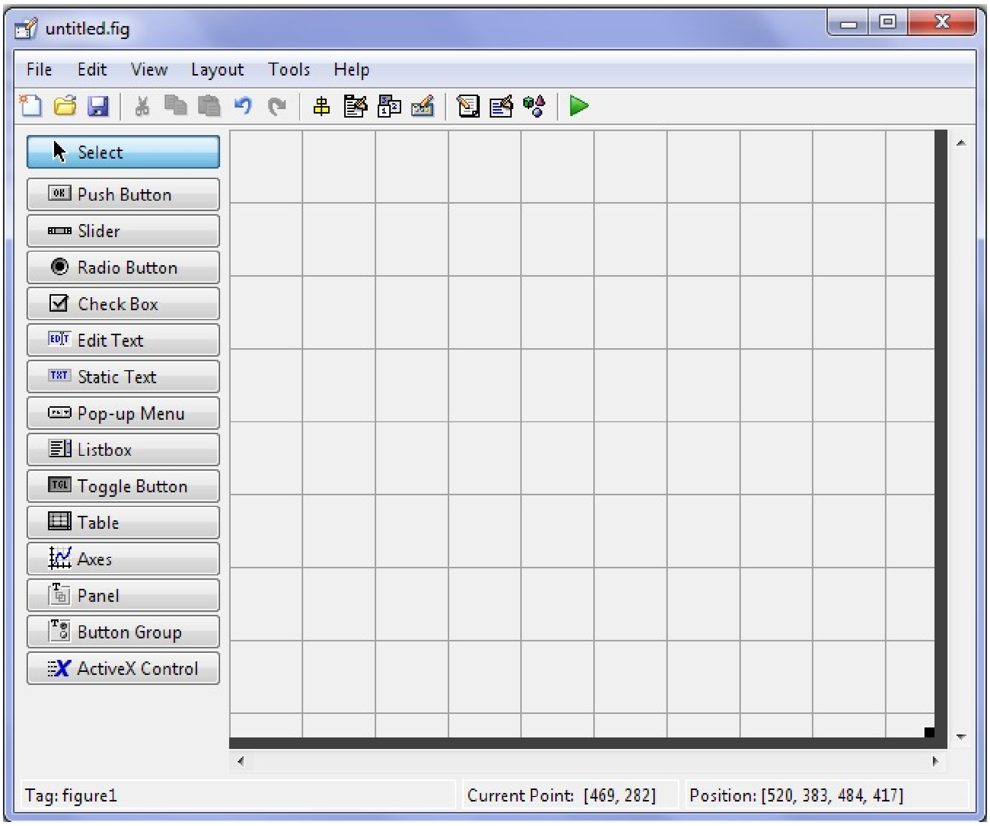


Fig. 2. GUIDE design window

3. GUI Components

Some basic GUI components are

(a) Push Button
(b) Slider
(c) Radio Button
(d) Check Box
(e) Edit Text
(f) Static Text
(g) Pop-up Menu
(h) Listbox
(i) Toggle Button
(j) Table
(k) Axes
(l) Panel
(m) Button Group
(n) ActiveX Control

These components may vary depending on the version of MATLAB. These examples are taken from MATLAB 2009a version.

4. A Simple Calculator

The target of this chapter is to give the reader a quick look at GUI of MATLAB rather than discussing each and every item. Readers will be able to learn with examples. The first example here will be considered to build a simple calculator. Two numbers provided by user will be added, subtracted, multiplied, and divided. The result will be displayed in a box. The first step of building a GUI is to have a rough sketch of the GUI. To build a simple calculator some basic components are needed:

(a) Two input boxes (Edit text)
(b) One output box (Static text)
(c) Four options for addition, subtraction, multiplication, and division (this can be accomplished in many ways. Here let's take four Radio Buttons)
(d) One Calculate button (Push Button)

To enhance the GUI one also can add some static texts to show various signs. In this example two more static texts are used. One is for the sign of calculation (+,-,x,/) and the other is to show (=) sign. The GUI will look like Fig. 3.

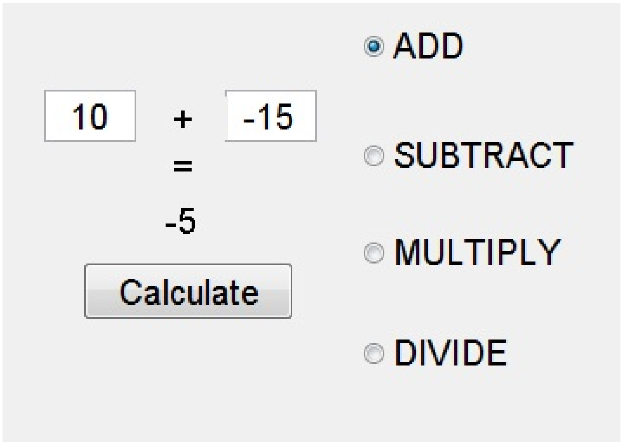
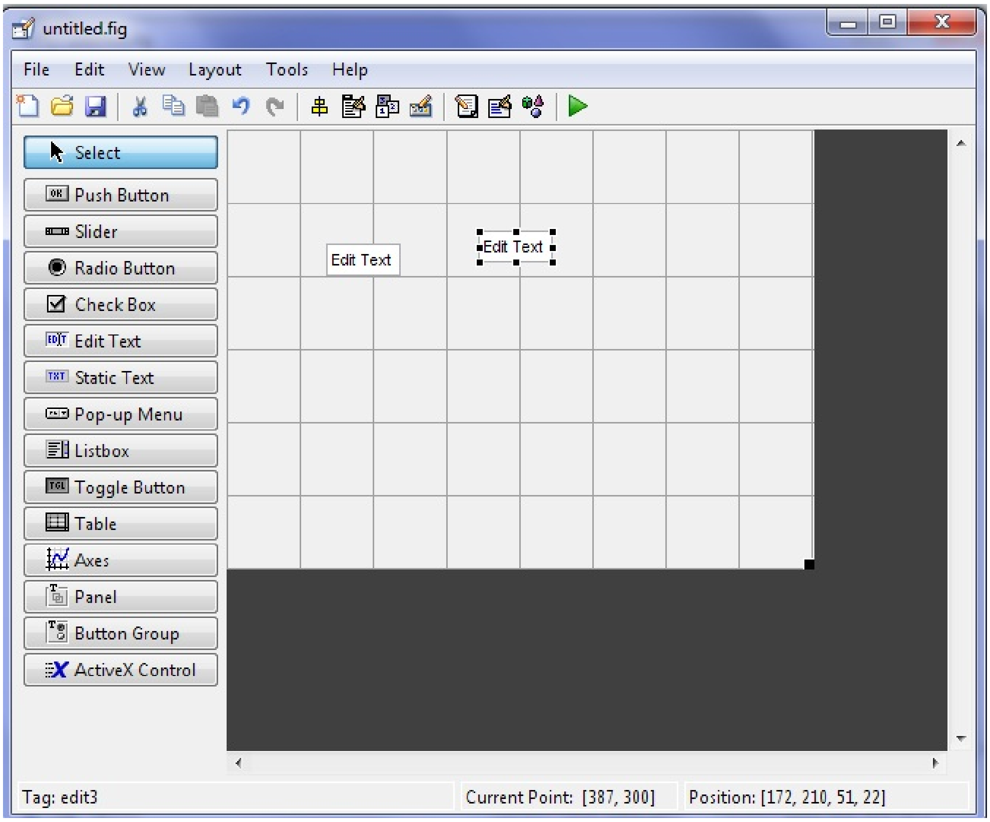


Fig. 3. A simple calculator

5. Adding Components

Now since the concept of the GUI has been built, the next step is to add all components required for building the calculators. Adding components to the *.fig file is very easy. Just click the item on the left, drag and drop to the blank space and the component is added. First consider adding the two static text components for entering two numbers. Select ***Edit Text*** and drag and drop to the blank space of the figure two times. Now the GUI will somewhat look like Fig. 4.



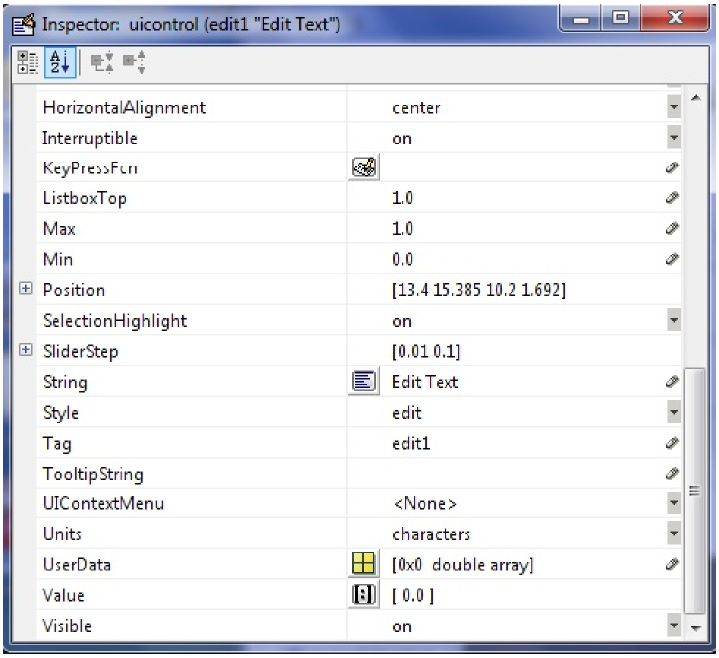
Tag: edit3 Current Point: [387, 300] Position: [172, 210, 51, 22]

Fig. 4. Adding edit text controls to the GUI

Now the newly added editable text boxes must be modified as per the need of the GUI. Here two numbers are added; hence it is a good idea to give two numbers as input from the very first so the inexperienced user will understand the purpose of the GUI. This task can be performed using Property Inspector.

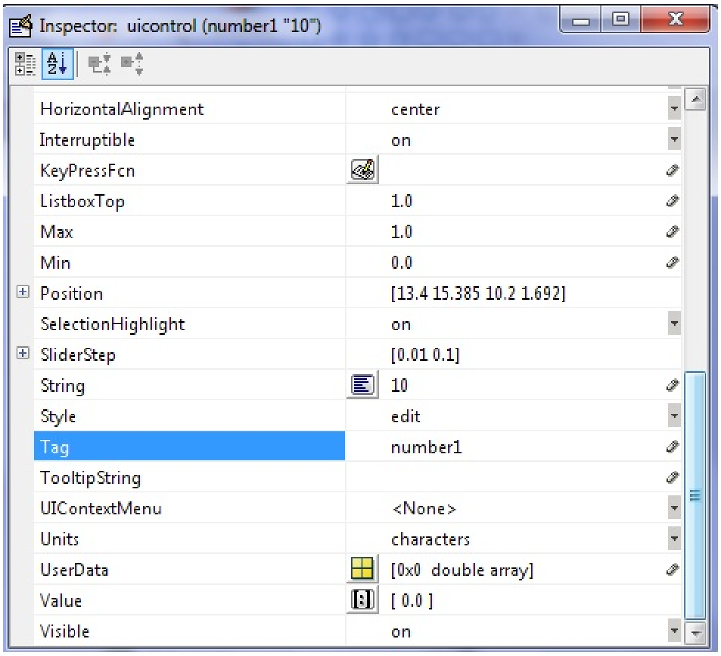
6. Property Inspector

Now right clicking on the component and selecting the Property Inspector will open the window as in Fig. 5. The left column gives the property name and the right column shows property values. String and Tag properties worth emphasizing since it is essential for programming a GUI. The String property has value Edit Text. Anything can be written here. Lets write *10* here and change the Tag to *number1*. Now the property inspector should look like Fig. 6. Similarly, change the property of the second Edit text. It is changed as: String *-15* and Tag *number2*.


Inspector: uicontrol (edit1 "Edit Text")

HorizontalAlignment	center
Interruptible	on
KeyPressFcn	
ListboxTop	1.0
Max	1.0
Min	0.0
Position	[13.4 15.385 10.2 1.692]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Edit Text
Style	edit
Tag	edit1
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	[0x0 double array]
Value	[0.0]
Visible	on

Fig. 5. Property inspector

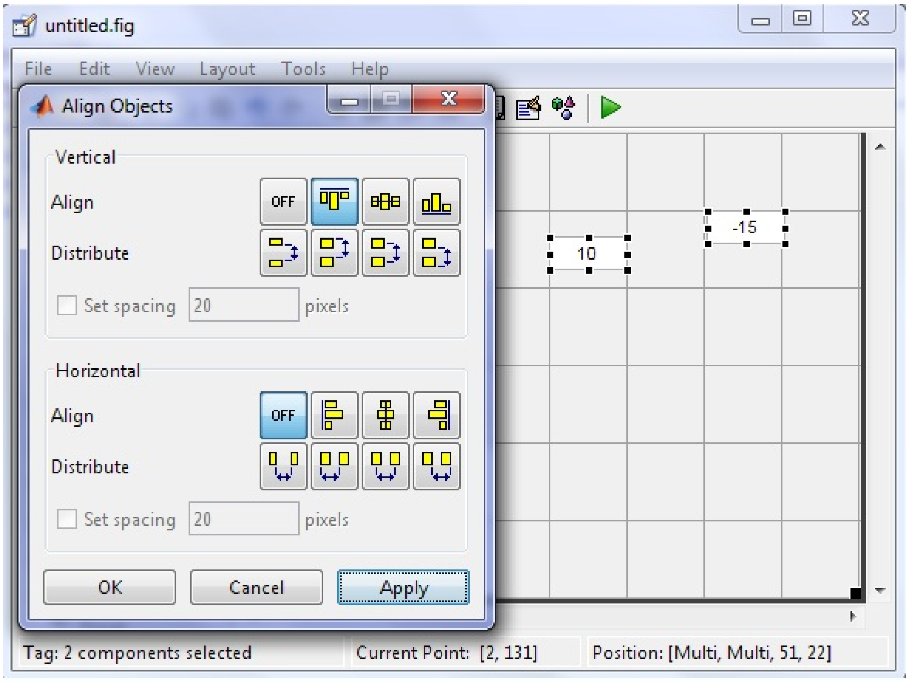

Inspector: uicontrol (number1 "10")

HorizontalAlignment	center
Interruptible	on
KeyPressFcn	
ListboxTop	1.0
Max	1.0
Min	0.0
Position	[13.4 15.385 10.2 1.692]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	10
Style	edit
Tag	number1
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	[0x0 double array]
Value	[0.0]
Visible	on

Fig. 6. Edited property inspector

7. Aligning Objects

One can align objects in the GUI to make the outlook of the GUI better. It can be done by clicking Tools menu and then selecting Align Objects. Then using the controls there objects can be aligned as in Fig. 7.



Set spacing 20 pixels

Set spacing 20 pixels

Tag: 2 components selected

Current Point: [2, 131]

Position: [Multi, Multi, 51, 22]

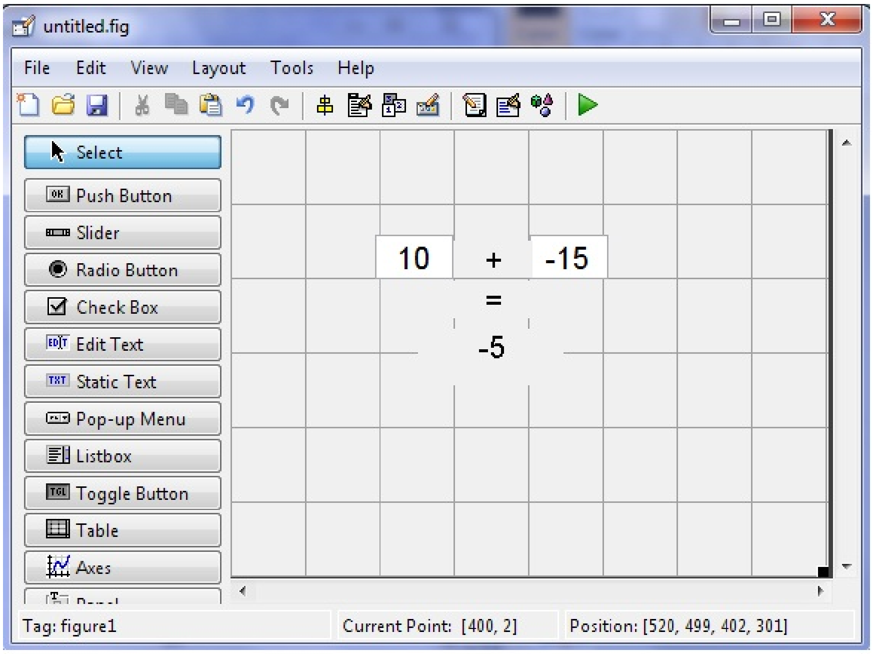
Fig. 7. Aligning objects

8. Adding More Components

More components can be added to the GUI. In Fig. 8 some Static Text controls are added and Fig. 9 shows the complete GUI with all components, Radio and Push Buttons are added here. The property modified for these components are given in Table 1.

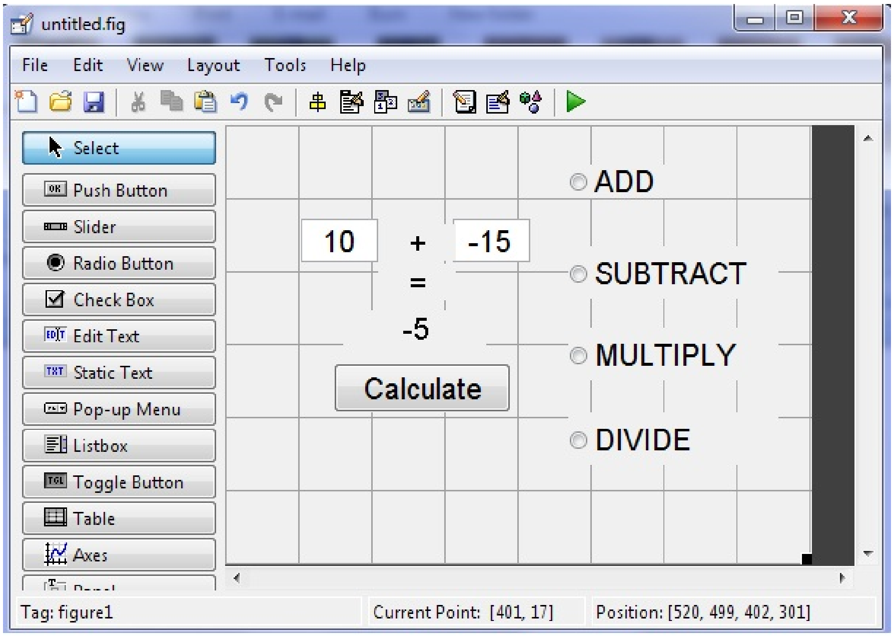
Component	FontSize	String	Tag
Edit Text1	15	10	edit1
Edit Text1	15	-15	edit2
Static Text1	15	+	text1
Static Text2	15	=	text2
Static Text3	15	-5	text3
Push Button	15	Calculate	pushbutton1
Radio Button1	15	ADD	radiobutton1
Radio Button2	15	SUBTRACT	radiobutton2
Radio Button3	15	MULTIPLY	radiobutton3
Radio Button4	15	DIVIDE	radiobutton4

Table 1. Properties of various controls used in this chapter



Tag: figure1 Current Point: [400, 2] Position: [520, 499, 402, 301]

Fig. 8. Adding static text to GUI



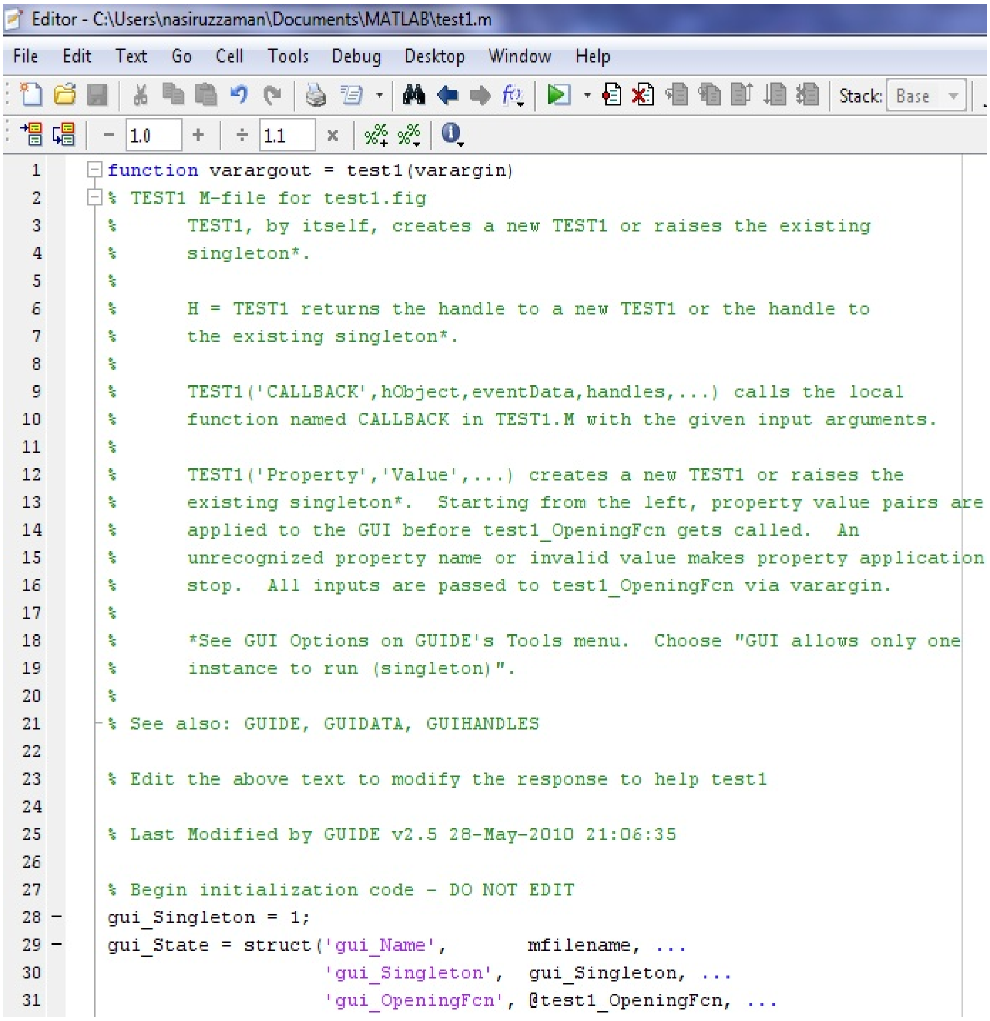
Tag: figure1 Current Point: [401, 17] Position: [520, 499, 402, 301]

Fig. 9. Complete figure of a simple calculator

9. Programming the GUI

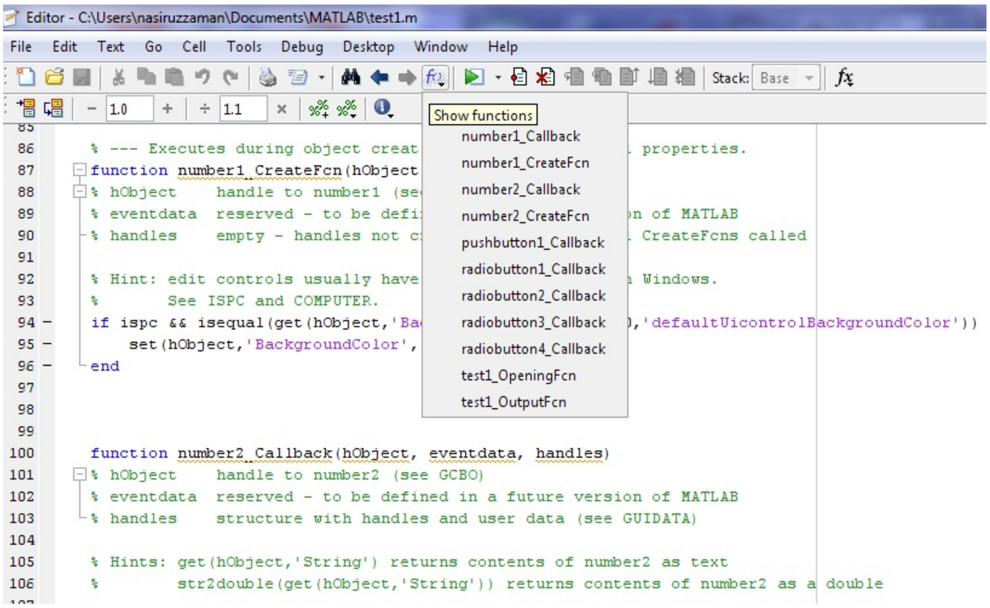
Now all the components are added. The GUI is saved in the name *test1.fig*. The rest task is coding the M-file. It can be accessed by clicking M-file Editor from View menu. When the m-file opens it somewhat looks like Fig. 10. This is a multi-function m-file. Codes are written under various functions. Functions are generated automatically. The opening function and callback functions are most important. Functions can be accessed as shown in Fig. 11.

When the GUI is first open the default action will be to work as adder. For this purpose let's modify the opening function of the GUI as in Fig. 12.


Editor - C:\Users\nasiruzzaman\Documents\MATLAB\test1.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1

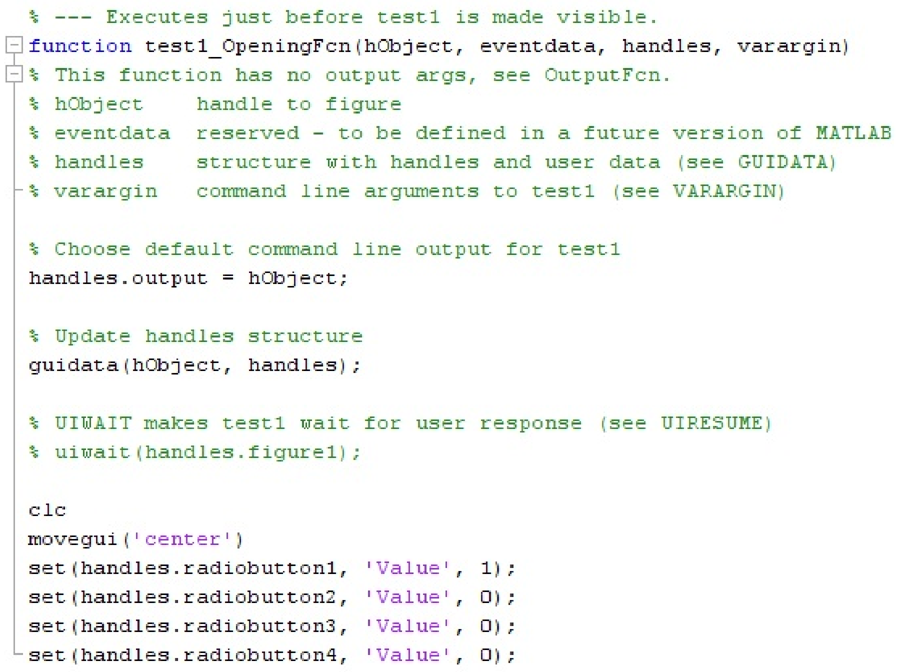
```
function varargout = test1(varargin)
% TEST1 M-file for test1.fig
%      TEST1, by itself, creates a new TEST1 or raises the existing
%      singleton*.
%
%      H = TEST1 returns the handle to a new TEST1 or the handle to
%      the existing singleton*.
%
%      TEST1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TEST1.M with the given input arguments.
%
%      TEST1('Property','Value',...) creates a new TEST1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before test1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to test1_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 test1

% Last Modified by GUIDE v2.5 28-May-2010 21:06:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @test1_OpeningFcn, ...
```

Fig. 10. M-file for simple calculator



```
    % --- Executes during object creat                                properties.
    function number1_CreateFcn(hObject
    % hObject    handle to number1 (se
    % eventdata  reserved - to be defi                                n of MATLAB
    % handles    empty - handles not c                                CreateFcns called

    % Hint: edit controls usually have                                Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'Ba                                ),'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',
    end



    function number2_Callback(hObject, eventdata, handles)
    % hObject    handle to number2 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of number2 as text
    %        str2double(get(hObject,'String')) returns contents of number2 as a double
```

Fig. 11. Accessing functions in *.m file

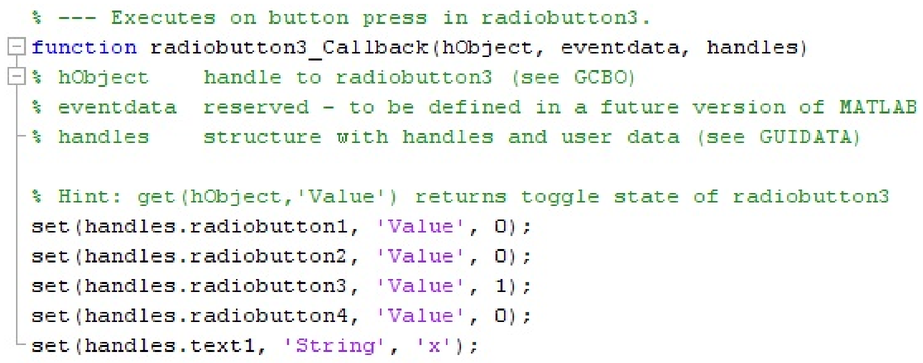


```
% --- Executes just before test1 is made visible.
function test1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to test1 (see VARARGIN)

% Choose default command line output for test1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes test1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

clc
movegui('center')
set(handles.radiobutton1, 'Value', 1);
set(handles.radiobutton2, 'Value', 0);
set(handles.radiobutton3, 'Value', 0);
set(handles.radiobutton4, 'Value', 0);
```

Fig. 12. Opening function of simple calculator

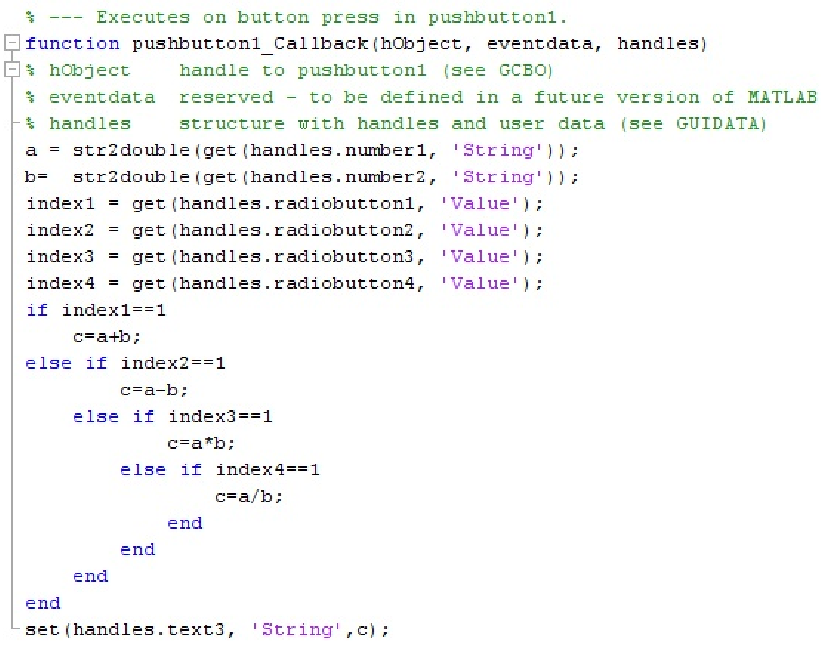
10. Programming Radio and Push Button

The radiobuttons should be mutually exclusive and when a radiobutton is selected corresponding operating notation should be reflected in the symbol text box. This job is done in the radiobutton callback function. One example (multiply radio button) is given in Fig. 13. The Calculate (Push) button is also programmed as in Fig. 14.



```
% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3
set(handles.radiobutton1, 'Value', 0);
set(handles.radiobutton2, 'Value', 0);
set(handles.radiobutton3, 'Value', 1);
set(handles.radiobutton4, 'Value', 0);
set(handles.text1, 'String', 'x');
```

Fig. 13. Radio Button callback



```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = str2double(get(handles.number1, 'String'));
b=  str2double(get(handles.number2, 'String'));
index1 = get(handles.radiobutton1, 'Value');
index2 = get(handles.radiobutton2, 'Value');
index3 = get(handles.radiobutton3, 'Value');
index4 = get(handles.radiobutton4, 'Value');
if index1==1
    c=a+b;
else if index2==1
        c=a-b;
    else if index3==1
            c=a*b;
        else if index4==1
                c=a/b;
            end
        end
    end
end
set(handles.text3, 'String',c);
```

Fig. 14. Push Button callback

11. Running the GUI

If the GUI is completed programming and run (whole code is given later) the window should look somewhat like Fig. 15 and if someone wants to multiply, the window will be like Fig. 16.

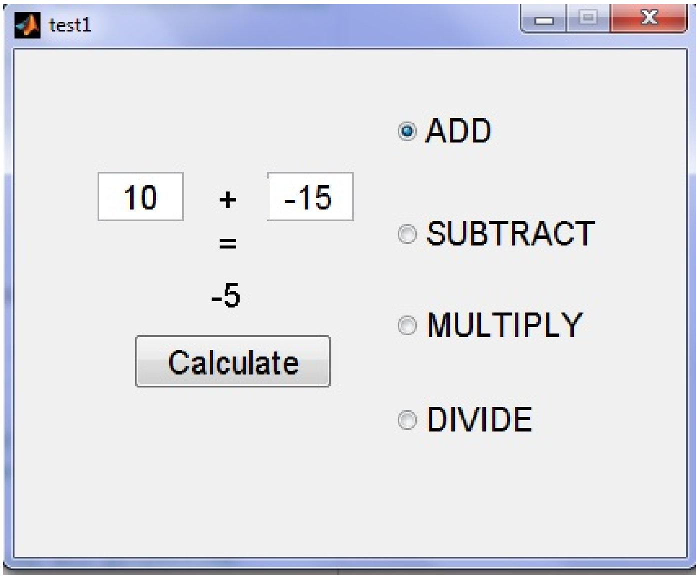


Fig. 15. Running GUI for the first time

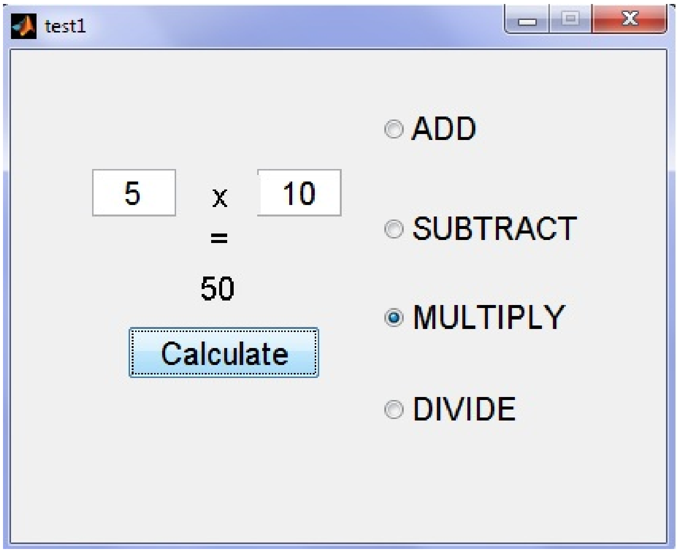
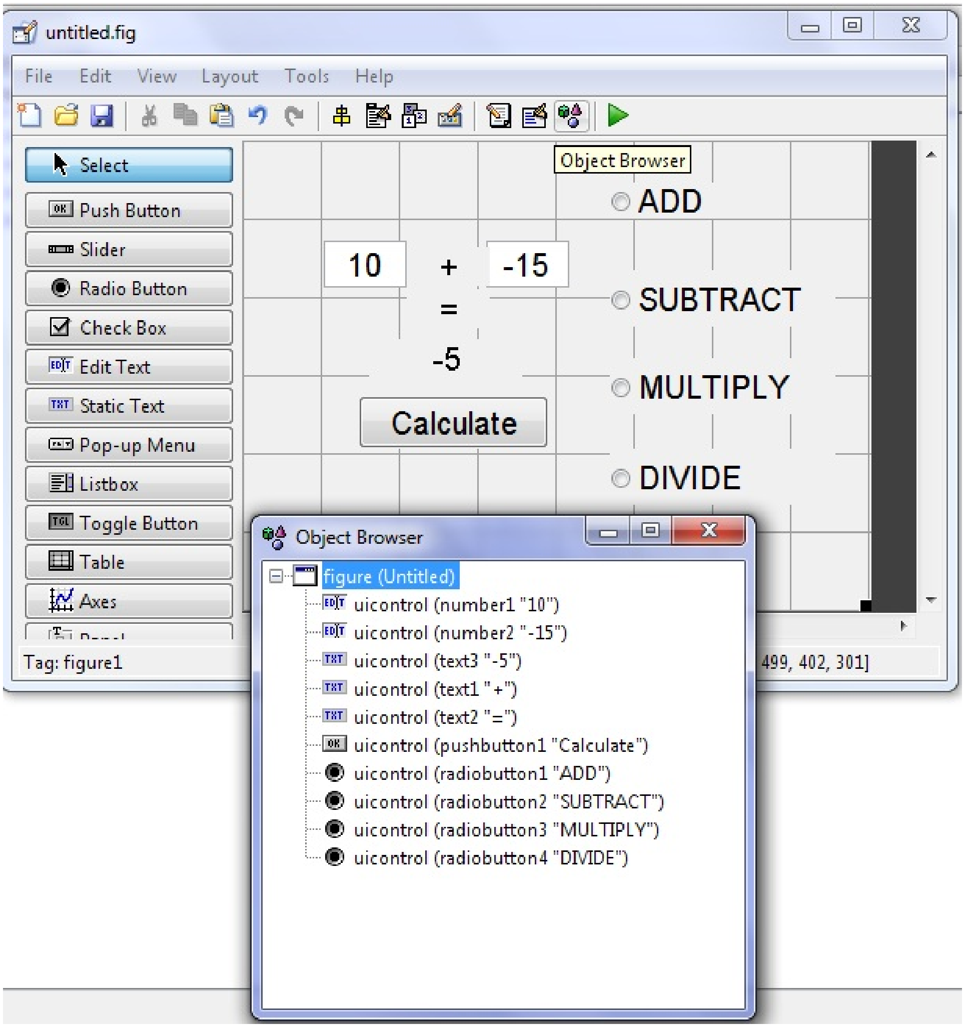


Fig. 16. Modifying and running GUI for multiplication

12. Some Additional Tools

There are also some additional options to make the GUI more attractive. Two examples are Object browser and Tab editor which are given in Fig. 17 and 18 respectively. The Object Browser displays a hierarchical list of the objects in the figure. It can be opened from View > Object Browser or by click the Object Browser icon on the GUIDE toolbar.

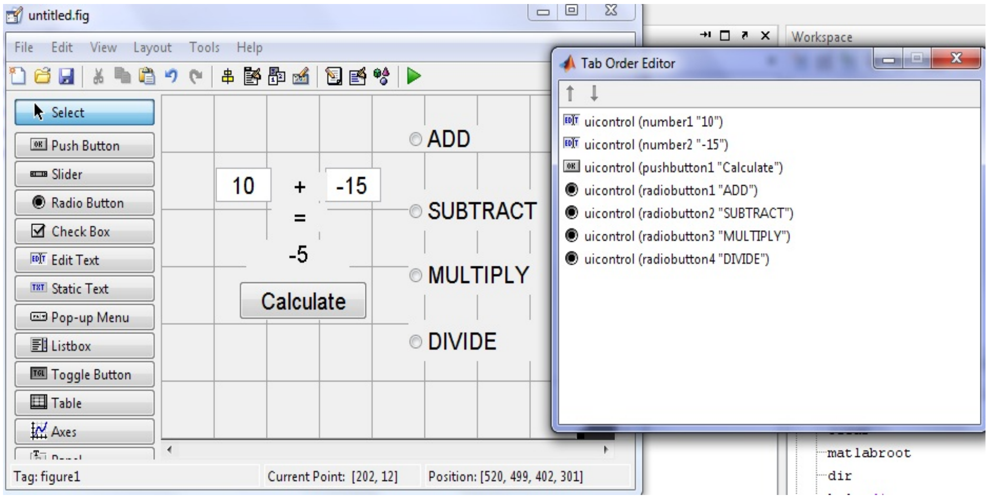


```
figure (Untitled)
    uicontrol (number1 "10")
    uicontrol (number2 "-15")
    uicontrol (text3 "-5")
    uicontrol (text1 "+")
    uicontrol (text2 "=")
    uicontrol (pushbutton1 "Calculate")
    uicontrol (radiobutton1 "ADD")
    uicontrol (radiobutton2 "SUBTRACT")
    uicontrol (radiobutton3 "MULTIPLY")
    uicontrol (radiobutton4 "DIVIDE")
```

Fig. 17. Object browser

A GUI's tab order is the order in which components of the GUI acquire focus when a user presses the Tab key on the keyboard. Focus is generally denoted by a border or a dotted border. To examine and change the tab order of the panel components, click the panel background to select it, then select Tab Order Editor in the Tools menu of the Layout Editor.

The Tab Order Editor displays the panel's components in their current tab order. To change the tab order, select a component and press the up or down arrow to move the component up or down in the list.



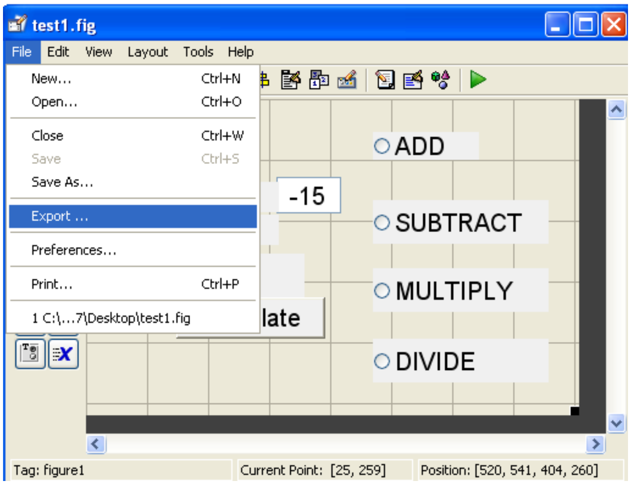
uicontrol (number1 "10")
uicontrol (number2 "-15")
uicontrol (pushbutton1 "Calculate")
uicontrol (radiobutton1 "ADD")
uicontrol (radiobutton2 "SUBTRACT")
uicontrol (radiobutton3 "MULTIPLY")
uicontrol (radiobutton4 "DIVIDE")

Tag: figure1 Current Point: [202, 12] Position: [520, 499, 402, 301]

Fig. 18. Tab editor

13. Running GUI from a Single *.m File

Up to this point to run a GUI, both *.fig and *.m files are required. A GUI can also run from a single *.m file which will be demonstrated here. At first go to the test1.fig file in the GUIDE and select Export from the File menu as depicted in Fig. 19. Let's save the GUI in the name *test_standard.m*.

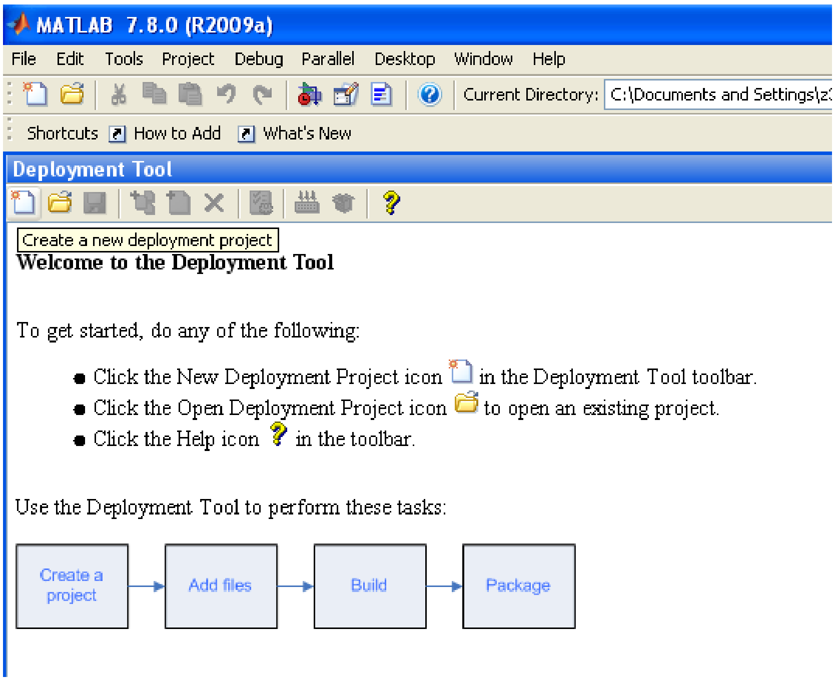


Tag: figure1 Current Point: [25, 259] Position: [520, 541, 404, 260]

Fig. 19. Running GUI from a single *.m file

14. Standalone Application Project

In this final step, the standalone project will be developed. Enter deploytool in the MATLAB command window (It may be needed to setup the MATLAB compiler by entering mbuild –setup and following steps in the command window). The MATLAB development project window appears as in Fig. 20. Clicking the new deployment project icon as shown in Fig. 20 opens a window as in Fig. 21.


MATLAB 7.8.0 (R2009a)
File Edit Tools Project Debug Parallel Desktop Window Help
Current Directory: C:\Documents and Settings\z
Shortcuts How to Add What's New
Deployment Tool
Create a new deployment project

Welcome to the Deployment Tool

To get started, do any of the following:

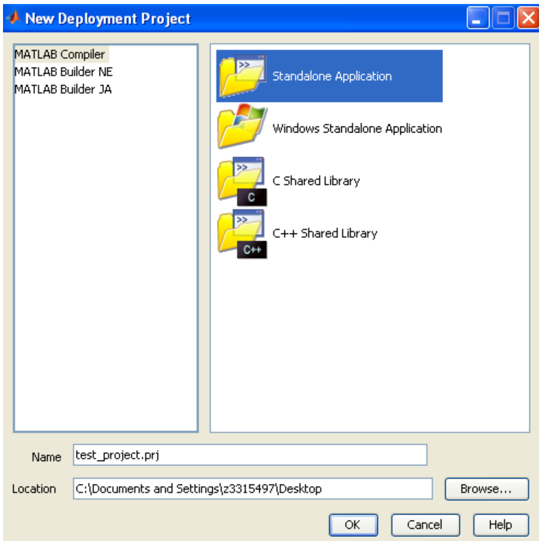
- Click the New Deployment Project icon in the Deployment Tool toolbar.
- Click the Open Deployment Project icon to open an existing project.
- Click the Help icon in the toolbar.

Use the Deployment Tool to perform these tasks:

Create a project
Add files
Build
Package

Fig. 20. Deployment tool

Here the project needs to be saved (let's save it by the name *test_project.prj*). After that, files should be added to the project (here *test_standard.m)* which is given in Fig. 22. The state of the deployment tool window after the file have been added is shown in Fig. 23. The next step is to build the project which is given in Fig. 24. After the compilation process the executable file will be available. The file can be found in the specified location in the ***distrib*** folder. In this project the file name should be *test_project.exe*. If the file is clicked in any PC it will run as in Fig. 25.



Name: test_project.prj

Location: C:\Documents and Settings\z3315497\Desktop

Fig. 21. New deployment project window

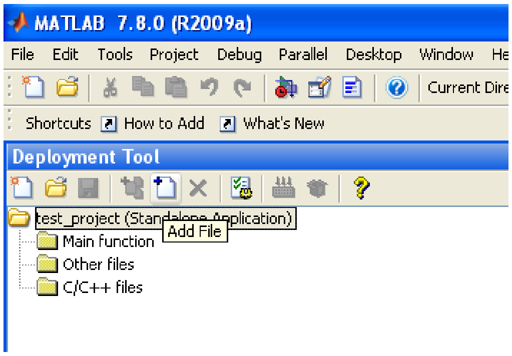

MATLAB 7.8.0 (R2009a)
File Edit Tools Project Debug Parallel Desktop Window He
Current Dire
Shortcuts How to Add What's New
Deployment Tool
test_project (Standalone Application)
Add File
Main function
Other files
C/C++ files

Fig. 22. Adding file to deployment tool

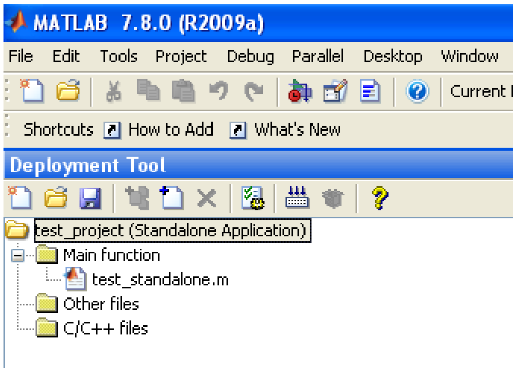

MATLAB 7.8.0 (R2009a)
File Edit Tools Project Debug Parallel Desktop Window
Current
Shortcuts How to Add What's New
Deployment Tool
test_project (Standalone Application)
Main function
test_standalone.m
Other files
C/C++ files

Fig. 23. Deployment tool window after adding file

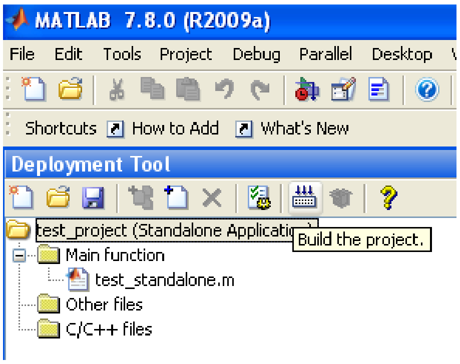

MATLAB 7.8.0 (R2009a)
File Edit Tools Project Debug Parallel Desktop
Shortcuts How to Add What's New
Deployment Tool
test_project (Standalone Applicatio
Build the project.
Main function
test_standalone.m
Other files
C/C++ files

Fig. 24. Build Project

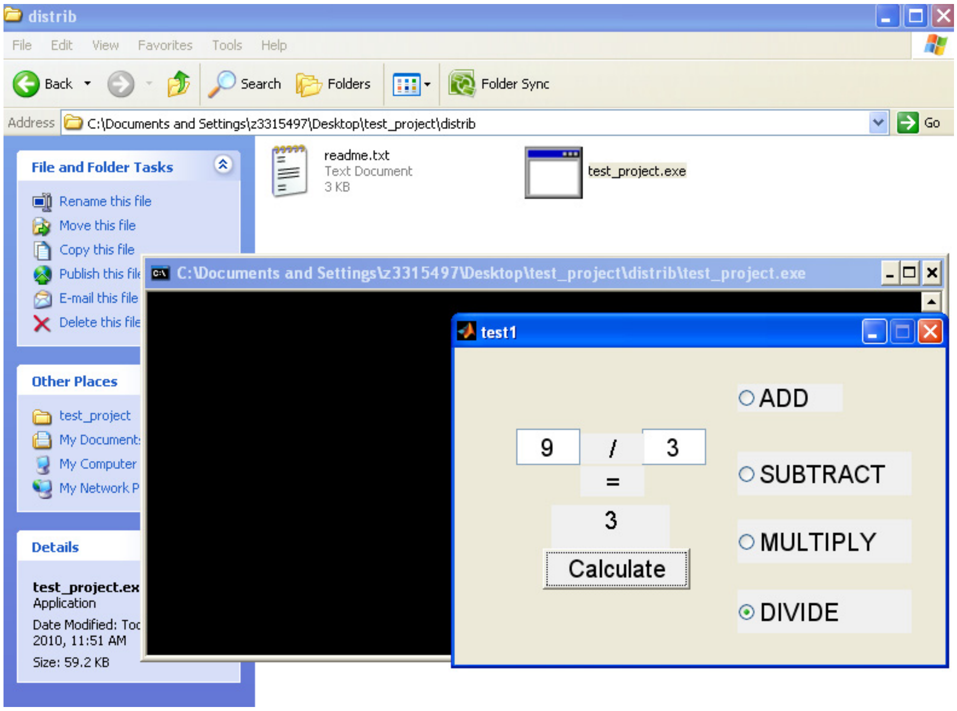


Fig. 25. Running the standalone project

15. Conclusion

A standalone MATLAB project is discussed here which will be very useful for educational purposes. Students can develop their projects in home and demonstrate in the class. Teachers can build excellent software packages in powerful computers and without can run it the classroom PCs with limited resource. There is no need of version compatibility, no need of huge memory requirement. After completing the project in this chapter it will open a new horizon for MATLAB users. For first time users codes are given in the next article. In case of any question regarding this issue the author can be contacted at nasiruzzaman@ieee.org.

16. MATLAB Code

```
function varargout = test1(varargin)
% TEST1 M-file for test1.fig
%      TEST1, by itself, creates a new TEST1 or raises the existing
%      singleton*.
%
%      H = TEST1 returns the handle to a new TEST1 or the handle to
%      the existing singleton*.
%
%      TEST1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TEST1.M with the given input arguments.
%
%      TEST1('Property','Value',...) creates a new TEST1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before test1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to test1_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 test1

% Last Modified by GUIDE v2.5 28-May-2010 22:04:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @test1_OpeningFcn, ...
                   'gui_OutputFcn',  @test1_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 test1 is made visible.
function test1_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 test1 (see VARARGIN)

% Choose default command line output for test1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes test1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

clc
movegui('center')
set(handles.radiobutton1, 'Value', 1);
set(handles.radiobutton2, 'Value', 0);
set(handles.radiobutton3, 'Value', 0);
set(handles.radiobutton4, 'Value', 0);


% --- Outputs from this function are returned to the command line.
function varargout = test1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function number1_Callback(hObject, eventdata, handles)
% hObject    handle to number1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of number1 as text
%        str2double(get(hObject,'String')) returns contents of number1 as a double


% --- Executes during object creation, after setting all properties.
function number1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to number1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function number2_Callback(hObject, eventdata, handles)
% hObject    handle to number2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of number2 as text
%        str2double(get(hObject,'String')) returns contents of number2 as a double


% --- Executes during object creation, after setting all properties.
function number2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to number2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = str2double(get(handles.number1, 'String'));
b=  str2double(get(handles.number2, 'String'));
index1 = get(handles.radiobutton1, 'Value');
index2 = get(handles.radiobutton2, 'Value');
index3 = get(handles.radiobutton3, 'Value');
index4 = get(handles.radiobutton4, 'Value');
if index1==1
   c=a+b;
else if index2==1
      c=a-b;
   else if index3==1
         c=a*b;
      else if index4==1
            c=a/b;
         end
      end
   end
end
set(handles.text3, 'String',c);



% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1
set(handles.radiobutton1, 'Value', 1);
set(handles.radiobutton2, 'Value', 0);
set(handles.radiobutton3, 'Value', 0);
set(handles.radiobutton4, 'Value', 0);
set(handles.text1, 'String', '+');

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2
set(handles.radiobutton1, 'Value', 0);
set(handles.radiobutton2, 'Value', 1);
set(handles.radiobutton3, 'Value', 0);
set(handles.radiobutton4, 'Value', 0);
set(handles.text1, 'String', '-');


% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3
set(handles.radiobutton1, 'Value', 0);
set(handles.radiobutton2, 'Value', 0);
set(handles.radiobutton3, 'Value', 1);
set(handles.radiobutton4, 'Value', 0);
set(handles.text1, 'String', 'x');


% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton4
set(handles.radiobutton1, 'Value', 0);
set(handles.radiobutton2, 'Value', 0);
set(handles.radiobutton3, 'Value', 0);
set(handles.radiobutton4, 'Value', 1);
set(handles.text1, 'String', '/');


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

17. References

Chapman, S. J. (2007). *MATLAB Programming for Engineers,* (4th), Thomson Learning, 049524449X

Littlefield, B. L. & Hanselman, D, C. (2004). *Mastering MATLAB 7,* (1st), Prentice Hall, 0131430181

Marchand, P. (2002). *Graphics and GUIs with MATLAB,* (3rd), Chapman & Hall, 1584883200

MathWorks, Inc. (2009). *MATLAB® Creating Graphical User Interfaces,* The MathWorks, Inc, Natick, MA 01760-2098, USA

McMahon, D. (2007). *MATLAB Demystified,* (1st), McGraw-Hill Publishing, 0071485511

Smith, S. T. (2006). *MATLAB Advanced GUI Development,* (1st), Dog Ear Publishing, 1598581813