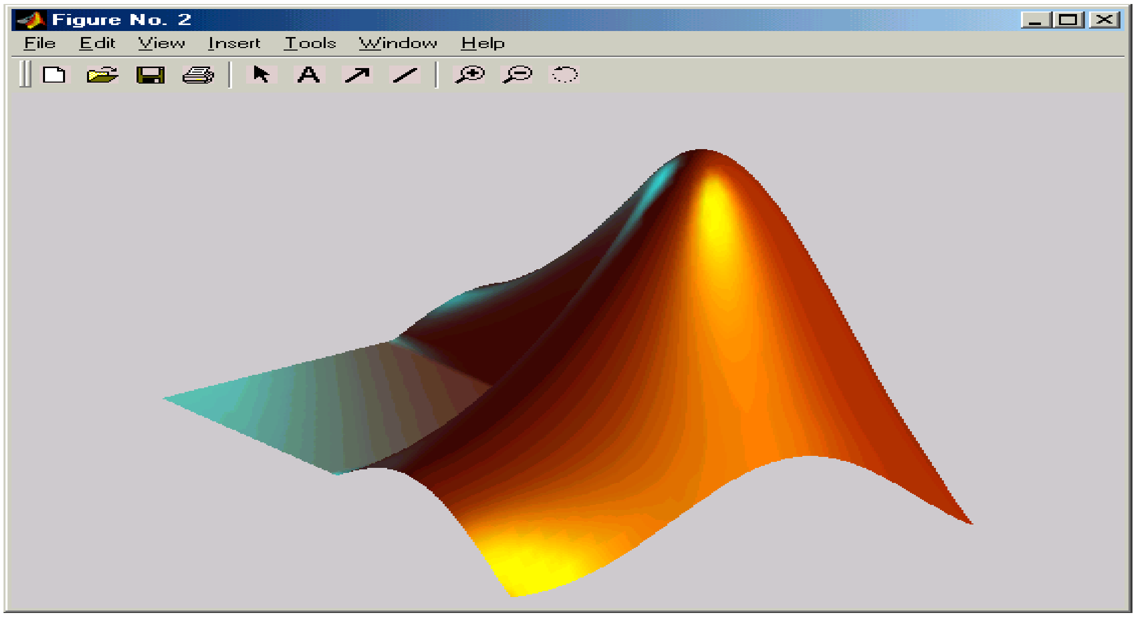
Matlab 6: GUI Tutorial.
Prepared by Vikram A Bose-Mullick.



Overview

Making Graphical User Interfaces in Matlab is very simple. A good place to begin learning about GUI development on the Matlab platform is to first understand how Matlab manages graphical objects. This particular tutorial focuses primarily on Matlab 6. This platform makes and excellent choice for developing interactive interfaces as the previous versions of Matlab had a noticeably clumsier and less mature feel when to came to developing GUI's.

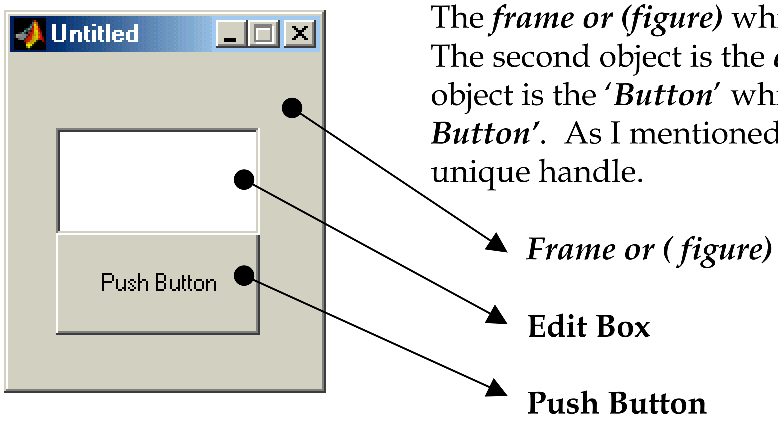
Developing GUI's on Matlab 6 is a breeze and hopefully this tutorial will be sufficient to get most anyone started. If you are running **an older version** of Matlab, this tutorial will help you get started however it will not be able to guide you all the way. I would recommend a migration to Matlab 6 as it as a more stable and a more mature platform where many of the bugs, especially in mat lab's ability to handle graphical objects have been addressed.

The main difference between Matlab 6 and the previous versions is the following. Matlab 6 uses java while the others used C++. I should mention that knowledge of neither platform is necessary to use matlab6 properly.

How does Matlab manages graphical objects?

Matlab is very similar to most other platforms, which allow GUI development. Any GUI in Matlab is in essence a collection of objects. Every object in the GUI has a unique handle (name). For instance, let's consider the following GUI. It is made up of three distinct objects, which are the following.

The ***frame or (figure)*** which is labeled '*Untitled*'. The second object is the ***edit box***. And the third object is the '***Button***' which is labeled ***'Push Button'***. As I mentioned every object must have an unique handle.

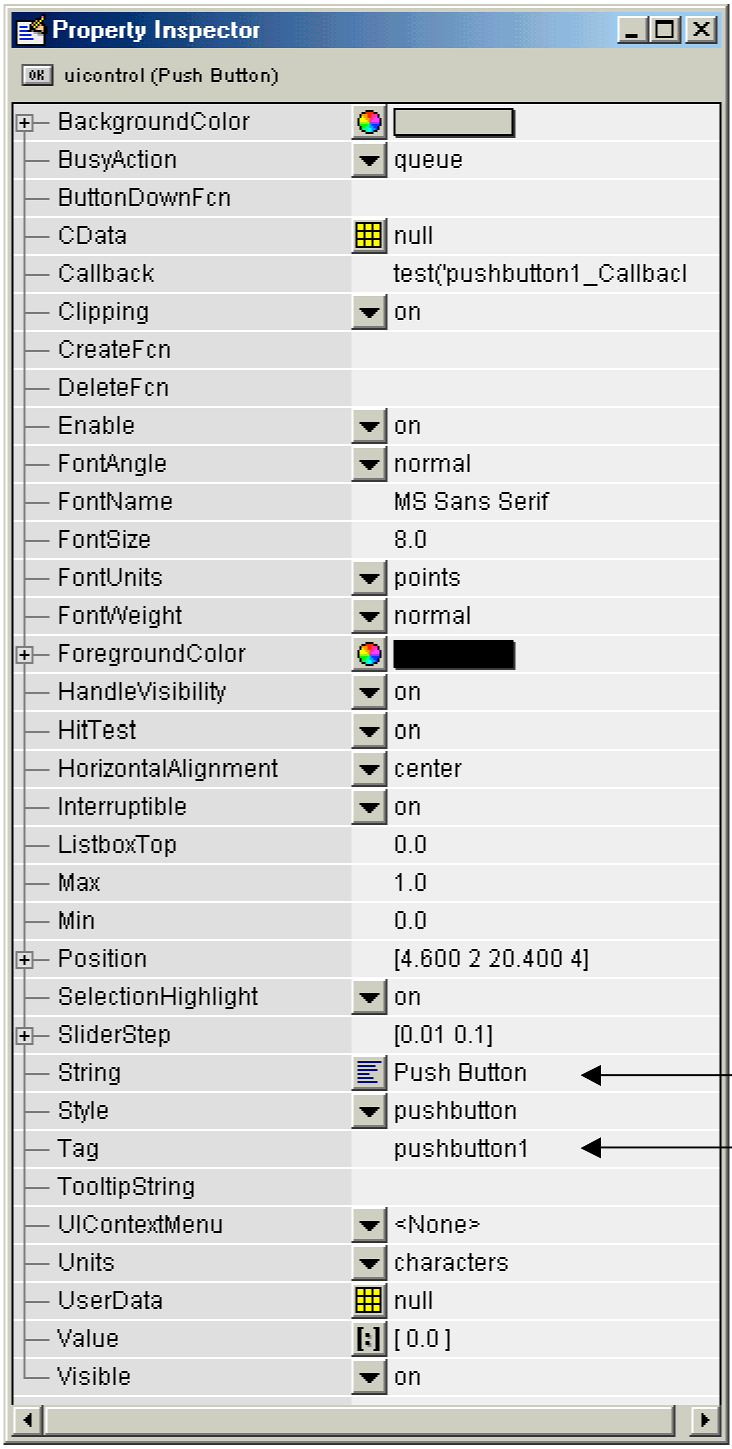


As I mentioned every object in the GUI will have an unique handle (name). This handle will allow developers to access the objects properties. ***For instance let's say we wanted to change the text on the button from 'Push Button' to 'Press ME'. All we have to do is obtain the object's handle. Once we have the handle we can then set the 'text' property of the object to 'Press ME'.*** Details regarding actually doing this will be discuss shortly, in this section of the tutorial I have deliberately decided to avoid code and focus on the overall process. The other properties of the object 'PushButton' include things like 'Background Color, Enable, Font, FontAngle, FontName, Position, String, Style, Tag, Value, etc. A full listing of it's properties are given below. The programmer can change any of these properties at design time.

Quick Re-Cap:

- GUI's are a collection of objects.
- Every object has a unique handle.
- Every object has properties.
- The user can alter these properties at design time.

Properties for the object known as 'Pushbutton1'



Property Inspector

uicontrol (Push Button)

BackgroundColor	⬜
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	test('pushbutton1_Callbacl
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	⬛
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0
Max	1.0
Min	0.0
Position	[4.600 2 20.400 4]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Push Button
Style	pushbutton
Tag	pushbutton1
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	null
Value	[0.0]
Visible	on

STRING PROPERTY

TAG PROPERTY

*Every thing about the object is listed here except for one very important property. This property is known as the **'Handle'**. The reason this property is not on the list is because the 'Handle' is assigned at Runtime. The user does not get to specify the handle however the user may obtain the handle in a number of different ways depending on the version of Matlab being used.*

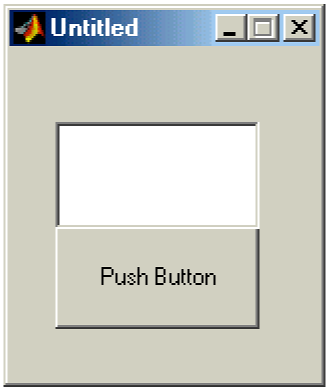
How do I obtain the Handle of an Object ?

There are a number of ways the user can obtain the handle of an object in Matlab. In Matlab 6 it is almost too easy. In previous versions of Matlab it's just a little bit more work. First we see how Matlab 6 manages handles.

IN MATLAB 6

Matlab 6 has a function, which collects every handle in the GUI and places it in a convenient data structure. This makes life very easy as the user does not have to poll every object for it's handle.

Example………



```
>> fig = openfig('test.fig');         %loads the figure.
>> handles = guihandles(fig)          %Gets handles.

handles =

      figure1: 102.0034
        edit1: 3.0043
  pushbutton1: 103.0039
```

IN PREVIOUS VERSION OF MATLAB

To obtain a handle in the previous version the user must poll the object for it's handle. To poll the object the user must give every GUI object a unique 'Tag'. For instance the default tag for the 'PushButton' is 'pustbutton1'. There for the following is what the user would need to do in order to obtain the handle of the object.

```
>> pushbutton1_handle = findobj('Tag','pushbutton1');
```

To obtain handle for the remaining objects in the GUI the user must poll every object individually for it's handle. If there are many objects on the GUI this process becomes laborious and tiresome.

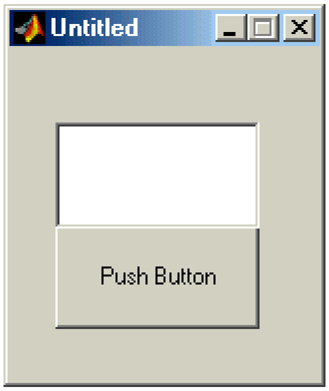
Once we have the 'Handle' how do we change properties ?

Matlab has two very important functions, which allow the user to alter an object's properties at their discretion. These functions are listed below.

GET Get object properties.
GET(H,'PropertyName') returns the value of the specified property for the graphics object with handle H.

SET Set object properties.
SET(H,'PropertyName',PropertyValue) sets the value of the specified property for the graphics object with handle H.

For instance consider the GUI discussed above, if we wanted to copy whatever was written on the button into the edit box we would need to do the following.

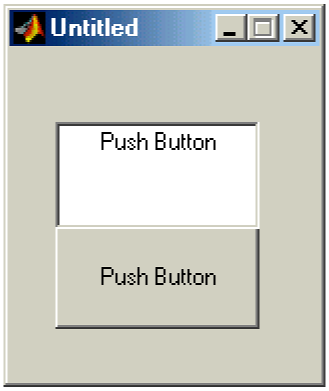


First we need the handles.

```
>> fig = openfig('test.fig');         %loads the figure.
>> handles = guihandles(fig)          %Gets handles.
```

Now we need to copy the 'String Property' of the Push Button into the Edit Box.

First we obtain the String Property of the Button by using the *get* function.



```
>>  var = get(handles.pushbutton1,'string');
```

The we put this data into the edit box by using the *set* function.

```
>> set(handles.edit1, 'string', var);
```

SUMMARY:

- **Guihandle() -> This function obtains all the handles in the GUI.**
- **Get() -> Allows users to obtain an object for a single property at runtime.**
- **Set() -> Allows users to change an objects property at runtime.**

EXERCISE 1: BUILDING A SIMPLE GUI IN MATLAB FROM SCRATCH.

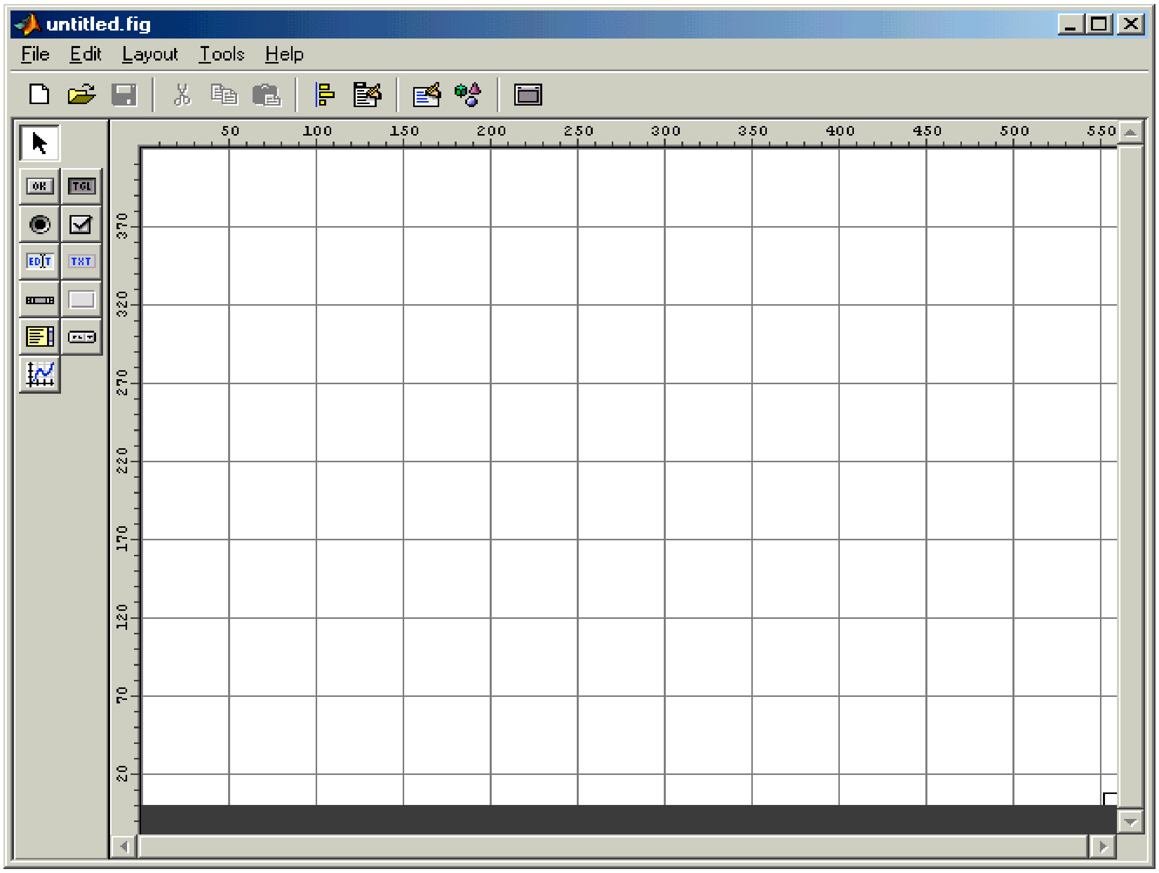
From this point in the tutorial we will take the shortest route to developing a fully functional GUI in Matlab. The previous section of the tutorial covered some of the concepts Matlab uses to manage graphical objects. Starting from this section the tutorial will become a lot more hands on.

Matlab is able to automatically generate a lot of code that is needed for GUI. When working under time constraints this feature of Matlab comes in very handy. For beginners, it's ok to rely on the automatically generated code.

THE GUIDE TOOL.

Matlab has a program called 'guide' which allows users to setup their GUI. The guide tool is very intuitive to use and can be accessed from the command line by simply typing in the following.

```
>> guide
```

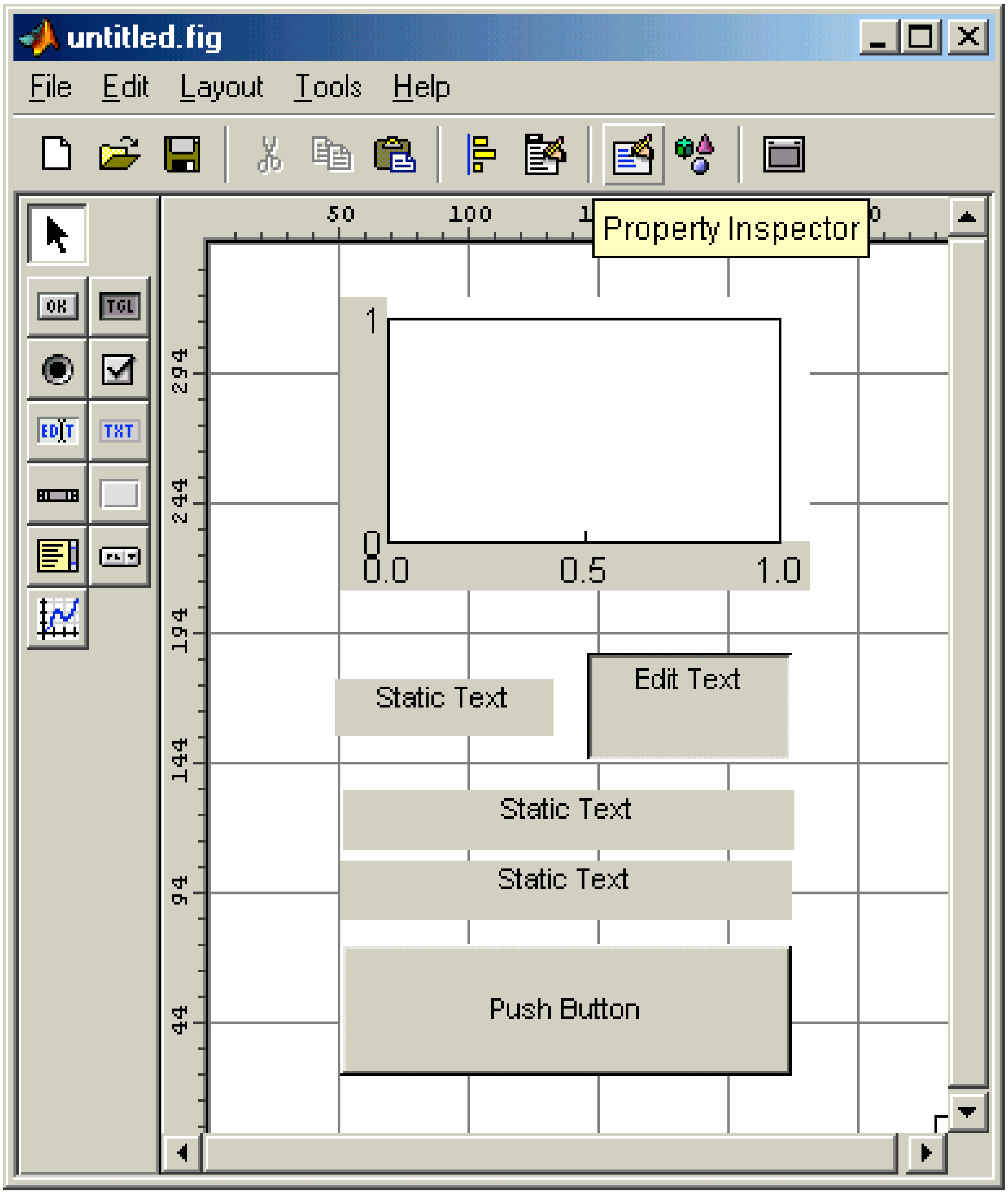


STEP 1: In this step we will setup our GUI.

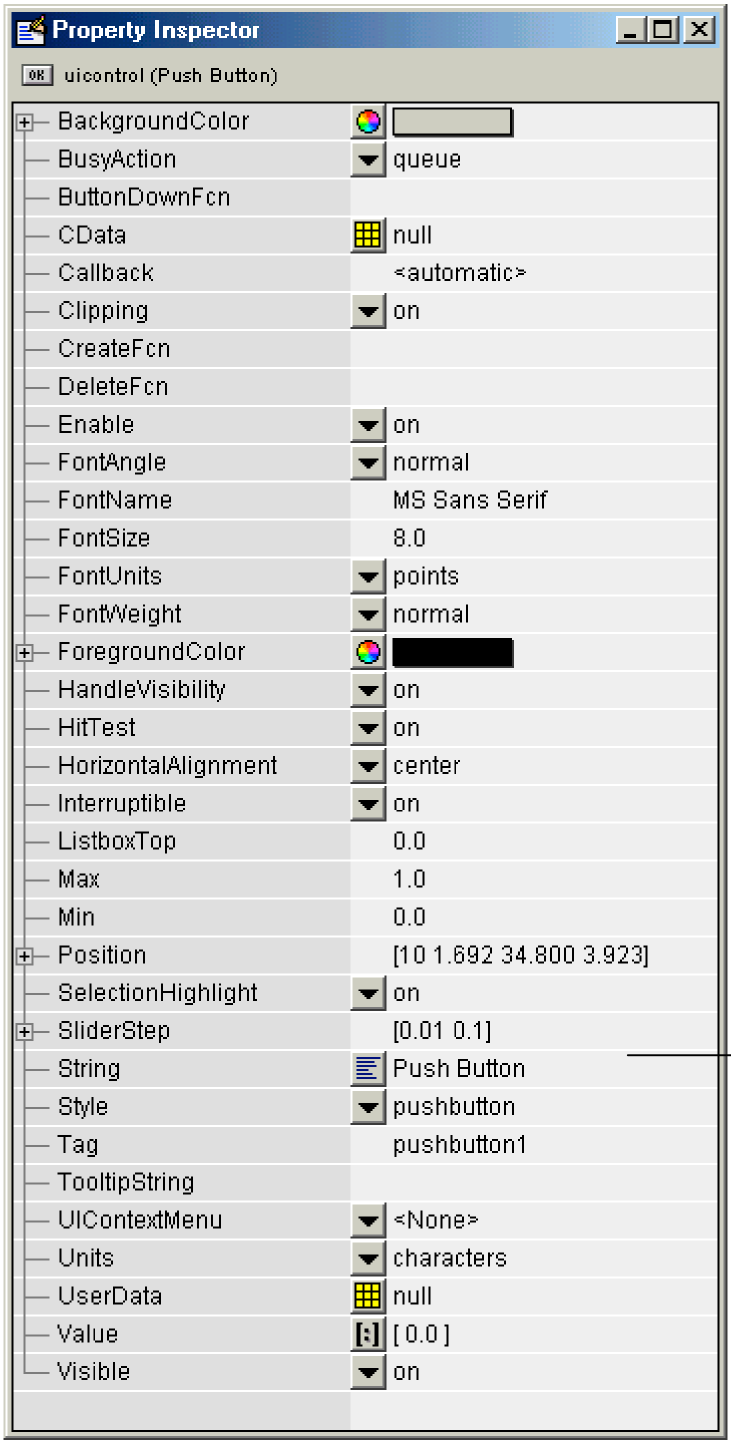
Start up Matlab6 and type in the following.

```
>> guide
```

STEP 2: Set up the GUI in the following manner.



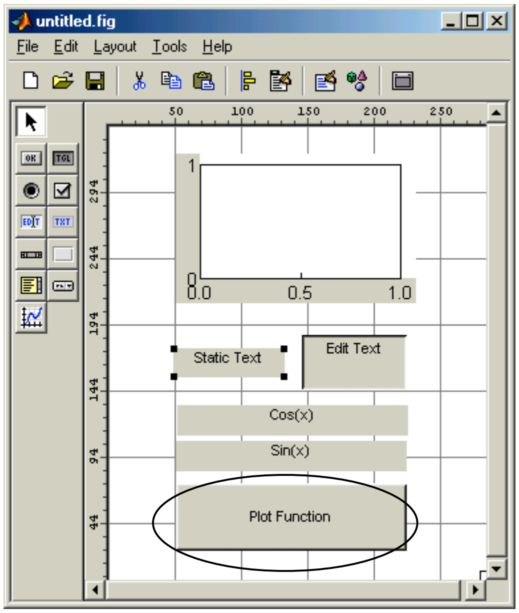
STEP 3: Right Click on the Push Button and a pop up menu should appear. From that menu pick the 'Inspect Properties' option.



Property Inspector

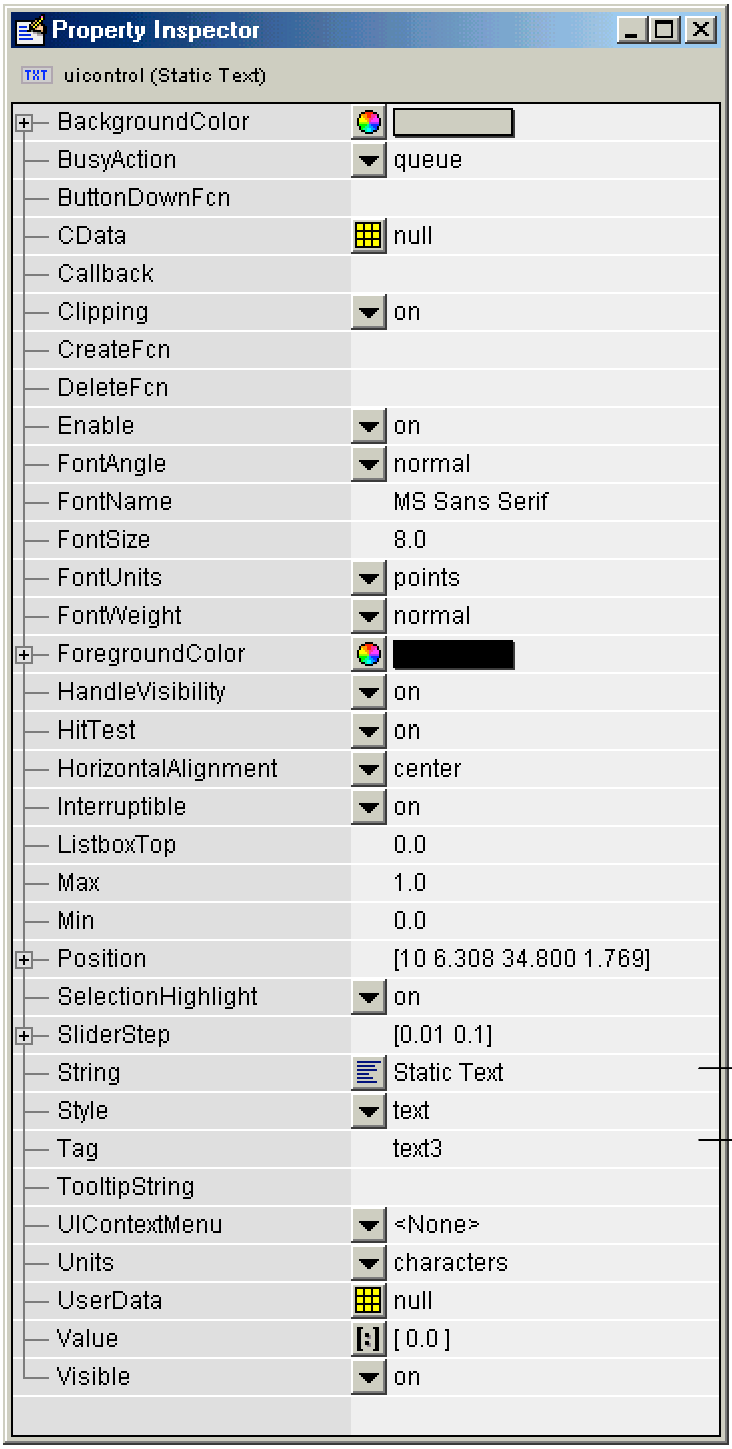
uicontrol (Push Button)

BackgroundColor	⬜
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	<automatic>
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	⬛
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0
Max	1.0
Min	0.0
Position	[10 1.692 34.800 3.923]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Push Button
Style	pushbutton
Tag	pushbutton1
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	null
Value	[0.0]
Visible	on



Change from 'Push Button' to 'Plot Function'.

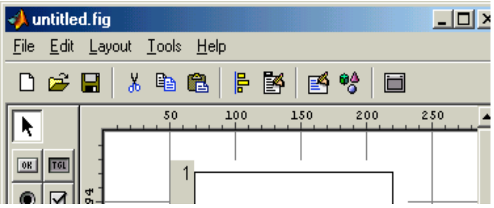
STEP 4: Right click on the lowest 'Static Text' object and select 'Inspect properties'.



Property Inspector

uicontrol (Static Text)

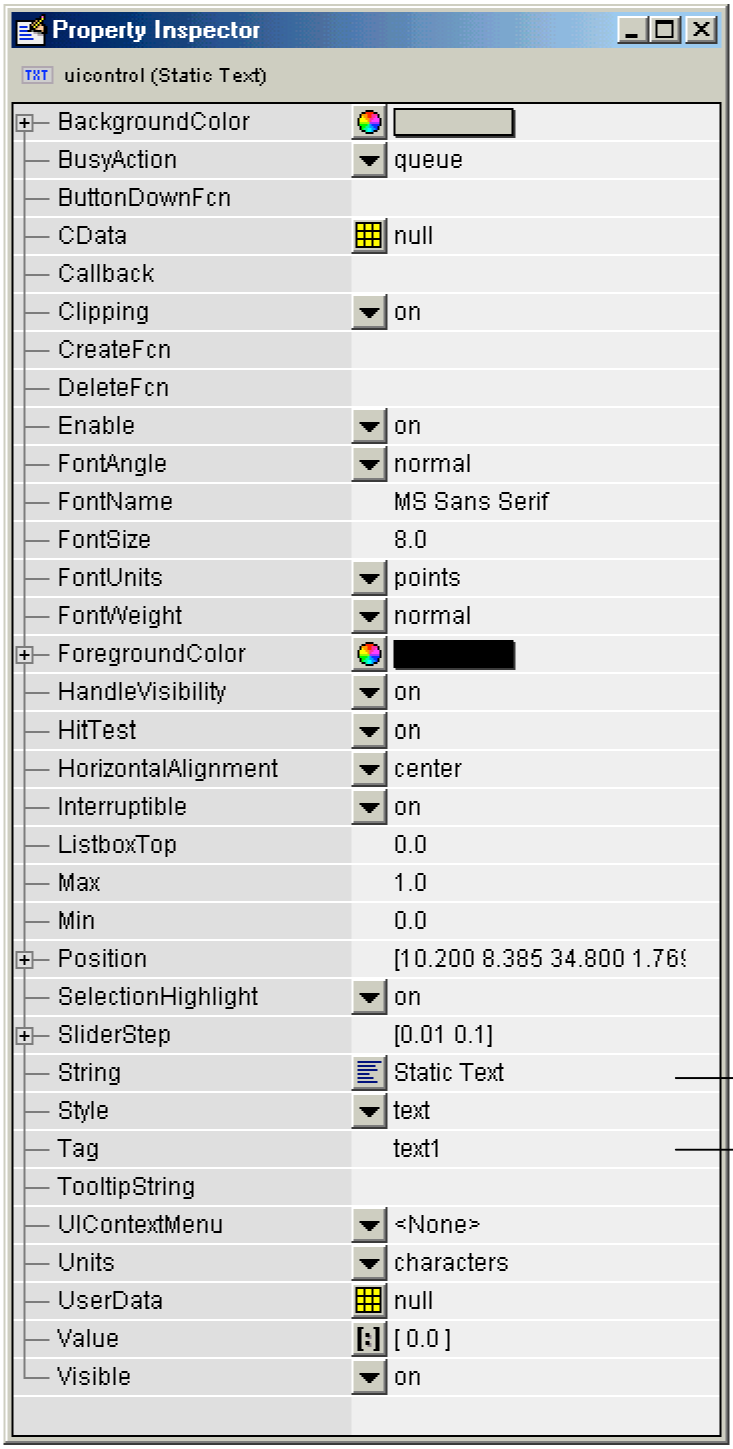
BackgroundColor	□
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	■
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0
Max	1.0
Min	0.0
Position	[10 6.308 34.800 1.769]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Static Text
Style	text
Tag	text3
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	null
Value	[0.0]
Visible	on



Change property from 'Static Text' to 'Sin(x)'.

Change property from 'text3' to 'sin_func'.

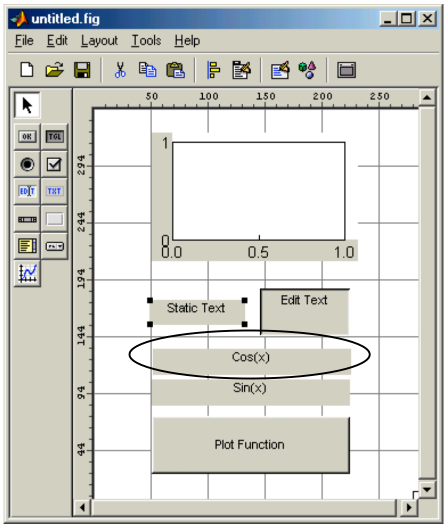
STEP 5: Right click on the other 'Static Text' object and select 'Inspect properties'.



Property Inspector

uicontrol (Static Text)

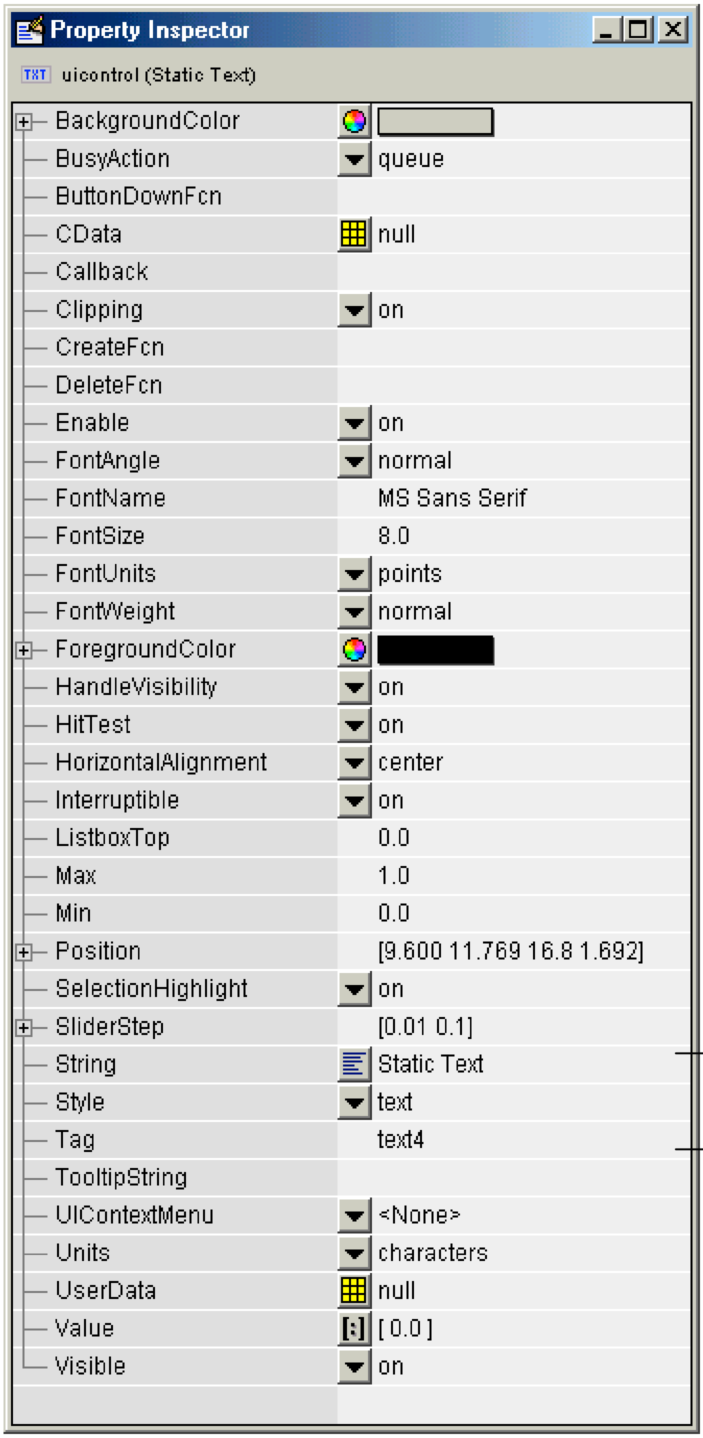
BackgroundColor	□
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	■
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0
Max	1.0
Min	0.0
Position	[10.200 8.385 34.800 1.76
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Static Text
Style	text
Tag	text1
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	null
Value	[0.0]
Visible	on



Change from 'Static Text' to 'Cos(x)'.

Change from 'text1' to 'cos_func'.

STEP 6: Right click on the third 'Static Text' object and select 'Inspect properties'.



Property Inspector

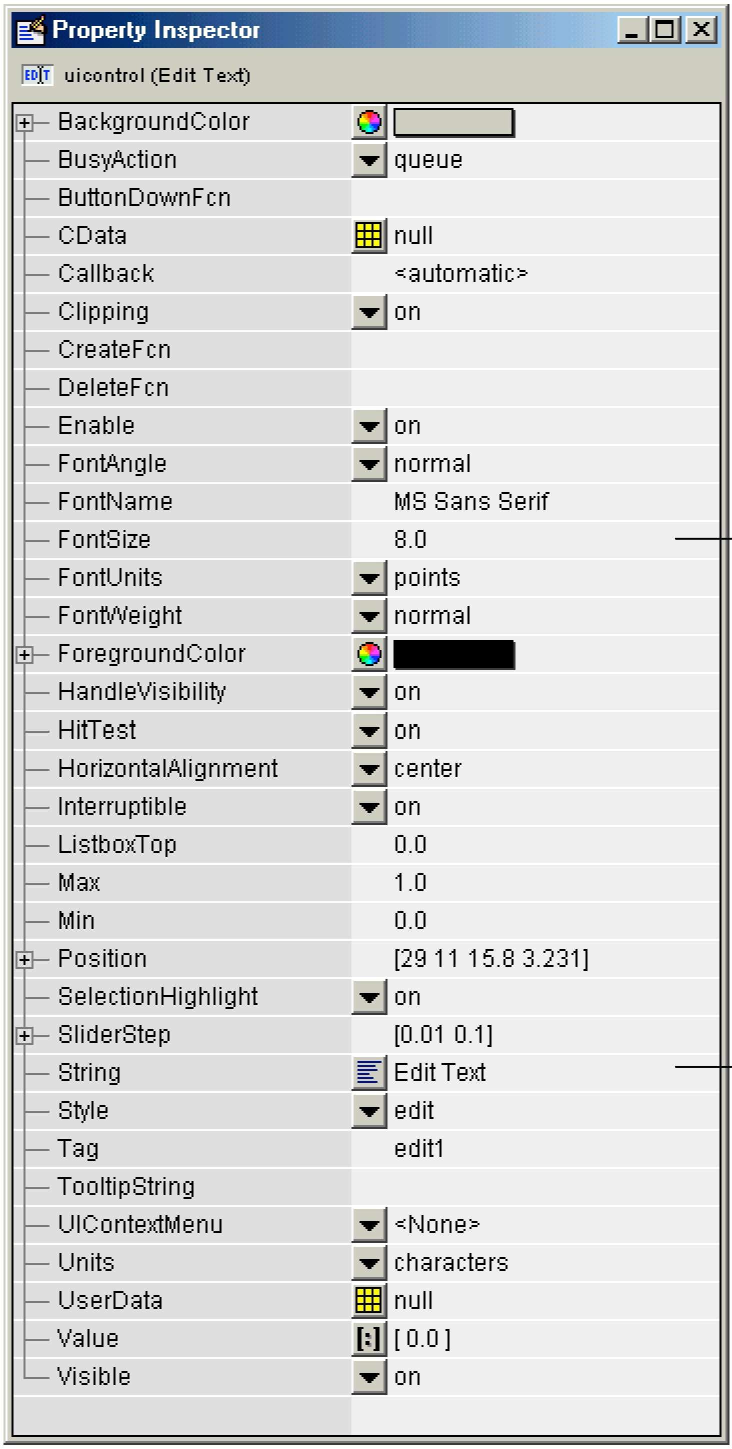
uicontrol (Static Text)

BackgroundColor	⬜
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	⬛
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0
Max	1.0
Min	0.0
Position	[9.600 11.769 16.8 1.692]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Static Text
Style	text
Tag	text4
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	null
Value	[0.0]
Visible	on

Change from 'Static Text' to 'Frequency'.

Change from 'text4' to 'freq'.

STEP 6: Right click on the third 'Edit Text' object and select 'Inspect properties'.



Property Inspector

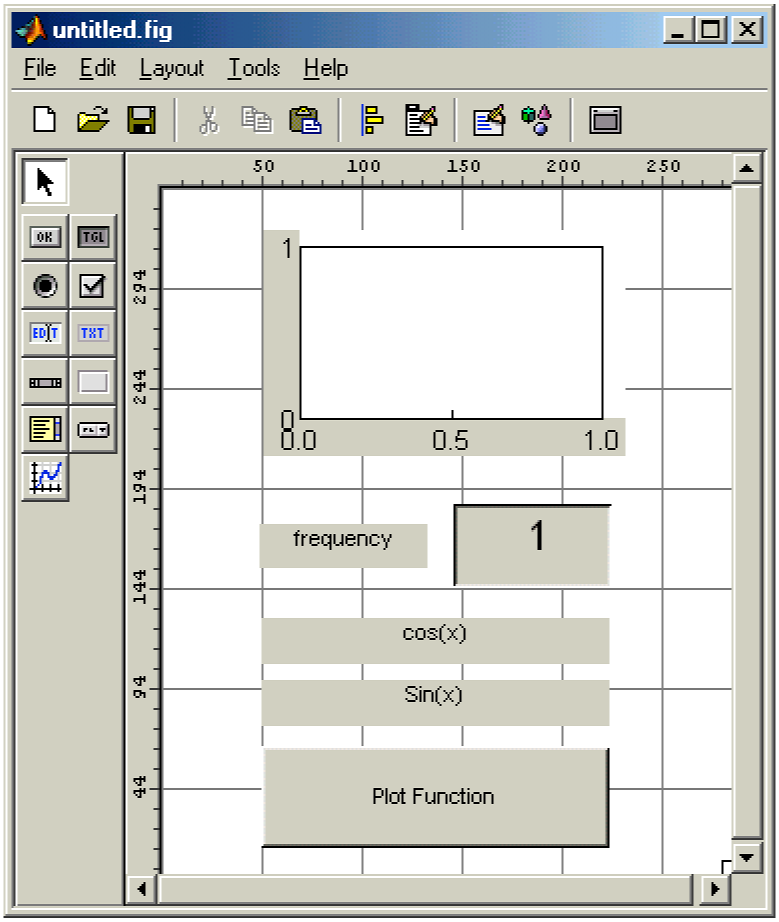
uicontrol (Edit Text)

BackgroundColor	⬜
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	<automatic>
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	⬛
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0
Max	1.0
Min	0.0
Position	[29 11 15.8 3.231]
SelectionHighlight	on
SliderStep	[0.01 0.1]
String	Edit Text
Style	edit
Tag	edit1
TooltipString	
UIContextMenu	<None>
Units	characters
UserData	null
Value	[0.0]
Visible	on

Change from '8.0' to '15'.

Change from 'Edit Text' to '1'.

STEP 7: At this point your GUI should look like this.

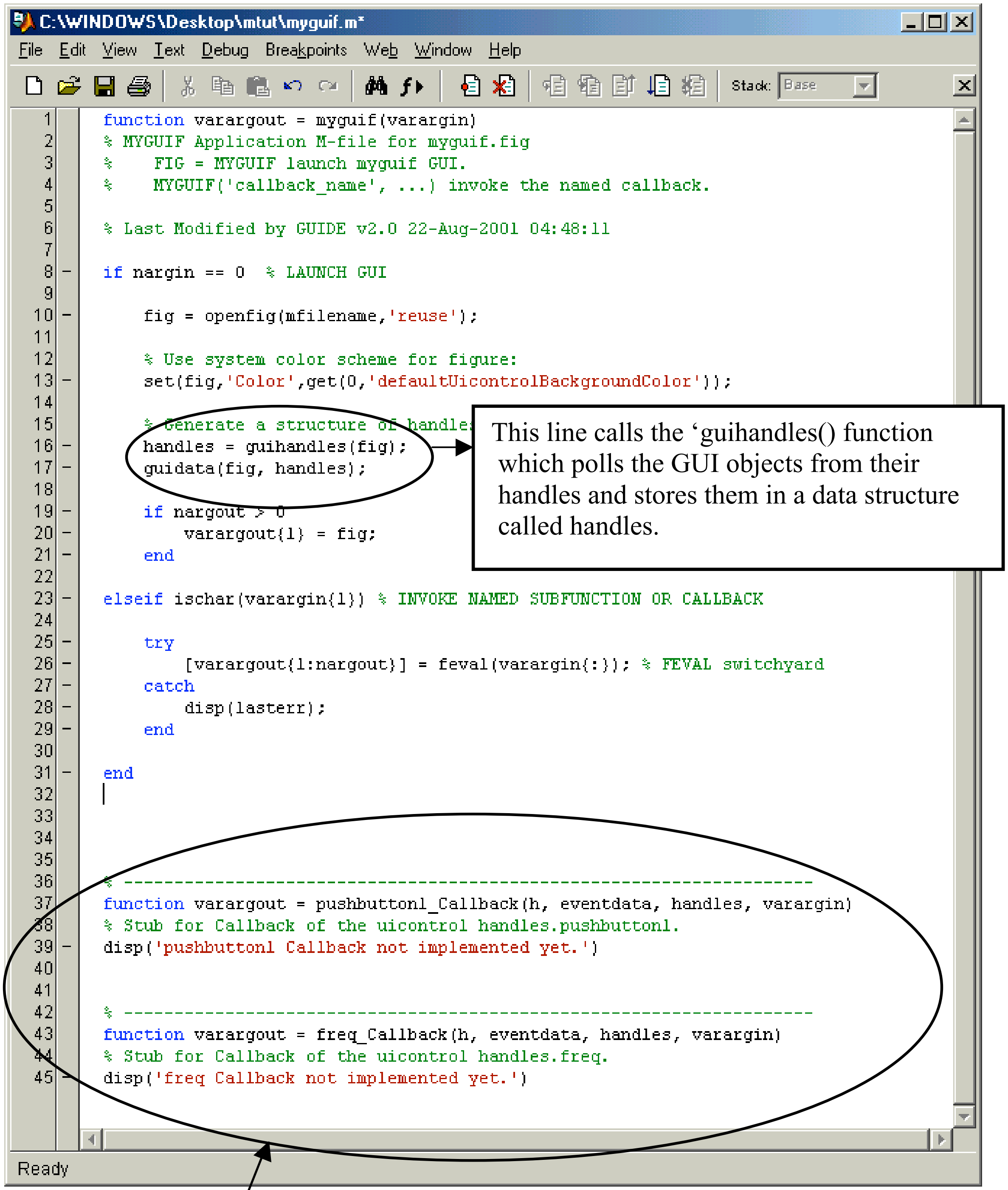


If you have made a mistake in the GUI you can easily correct it at this point without moving on. For instance, if you look closely in this GUI there is really no way to select which of the two (sin(x) or cos(x) are to be plotted. We should have used a different object in the first place.. perhaps a check box. We can easily delete the wrong object and replace it with the one we want. Since this is our first GUI we will keep it simple and get rid on one of the static text boxes.

Click on the Cos(x) box and hit the *delete* key and the cos(x) should disappear from the GUI. Once that is done, save as 'mygui'.

As soon as you save it Matlab should generate the skeleton source code for you and the source code should automatically open in an editor.

Understanding the Skeletal Code



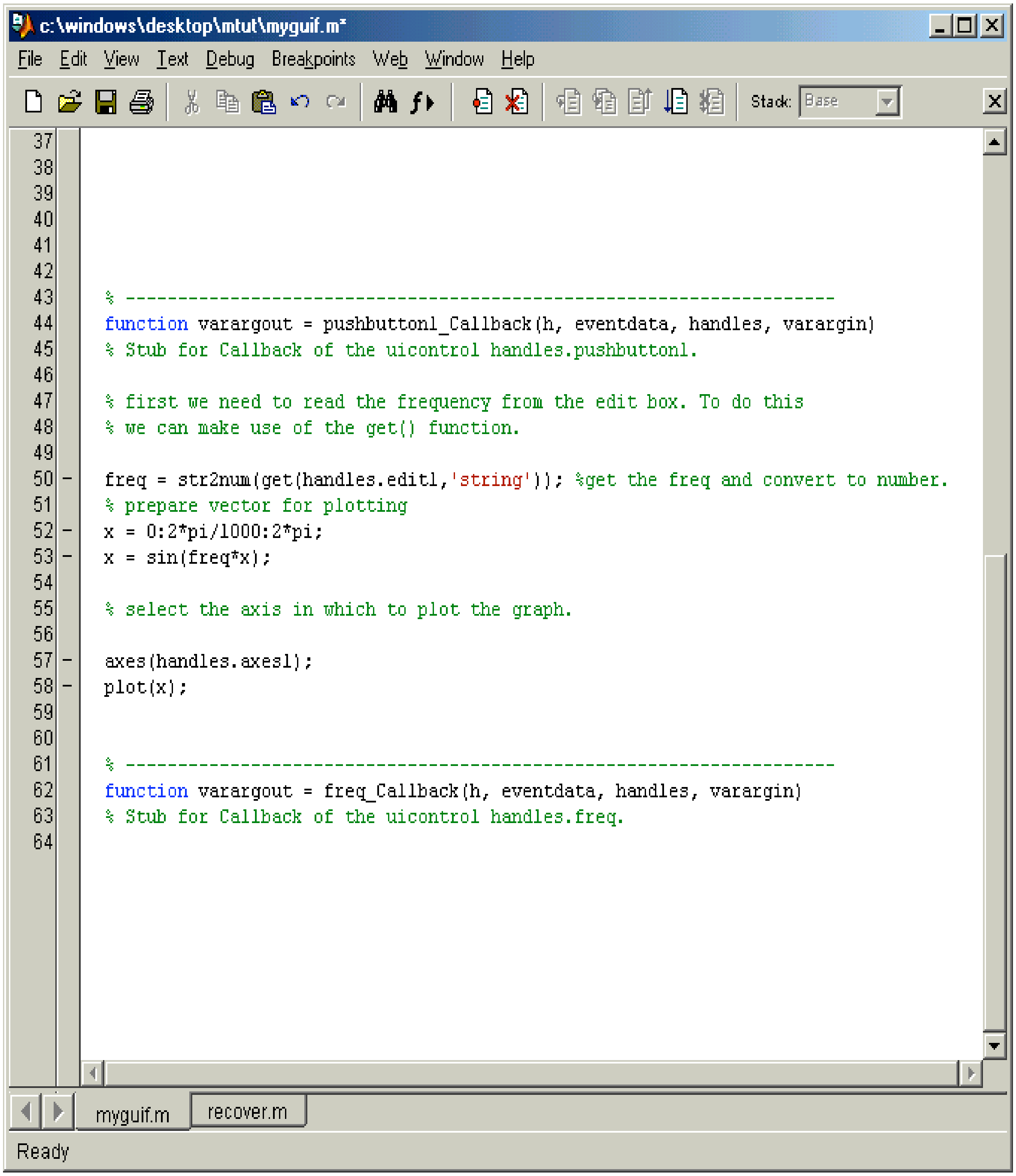
```
function varargout = myguif(varargin)
% MYGUIF Application M-file for myguif.fig
%    FIG = MYGUIF launch myguif GUI.
%    MYGUIF('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 22-Aug-2001 04:48:11

if nargin == 0  % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Use system color scheme for figure:
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles
    handles = guihandles(fig);
    guidata(fig, handles);

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
    catch
        disp(lasterr);
    end

end




% --------------------------------------------------------------------
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton1.
disp('pushbutton1 Callback not implemented yet.')


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

This line calls the ‘guihandles() function which polls the GUI objects from their handles and stores them in a data structure called handles.

These are stubs where objects such as buttons, edit boxes get their bodies.

Step 8: Activating the buttons.

Add the following code to the skeletal code.


c:\windows\desktop\mtut\myguif.m*
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
myguif.m
recover.m
Ready

```
% ----------------------------------------------------------------------
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton1.

% first we need to read the frequency from the edit box. To do this
% we can make use of the get() function.

freq = str2num(get(handles.edit1,'string')); %get the freq and convert to number.
% prepare vector for plotting
x = 0:2*pi/1000:2*pi;
x = sin(freq*x);

% select the axis in which to plot the graph.

axes(handles.axes1);
plot(x);


% ----------------------------------------------------------------------
function varargout = freq_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.freq.
```

STEP 9: Running the program.

To run the program, simply go to the Matlab main window and call your program.

```
>> mygui
```