

# blinkdagger

an Engineering and MATLAB blog

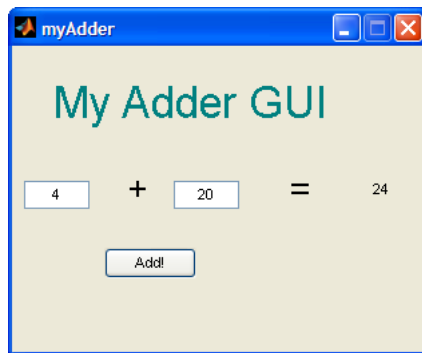
- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

## MATLAB GUI (Graphical User Interface) Tutorial for Beginners

23 Oct 2007 [Quan Quach](#) 341 comments 106,587 views



Why use a GUI in MATLAB? The main reason GUIs are used is because it makes things simple for the end-users of the program. If GUIs were not used, people would have to work from the command line interface, which can be extremely difficult and frustrating. Imagine if you had to input text commands to operate your web browser (yes, your web browser is a GUI too!). It wouldn't be very practical would it? In this tutorial, we will create a simple GUI that will add together two numbers, displaying the answer in a designated text field.



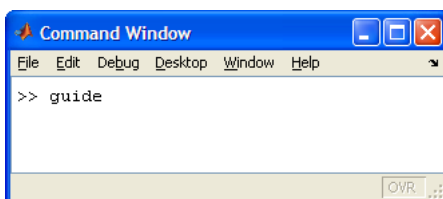
This tutorial is written for those with little or no experience creating a MATLAB GUI (Graphical User Interface). Basic knowledge of MATLAB is not required, but recommended. MATLAB version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Lets get started!

## Contents

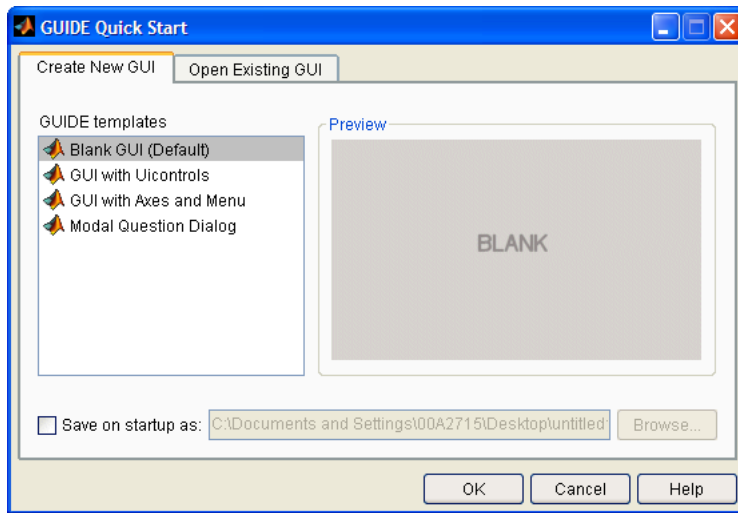
- [Initializing GUIDE \(GUI Creator\)](#)
- [Creating the Visual Aspect of the GUI: Part 1](#)
- [Creating the Visual Aspect of the GUI: Part 2](#)
- [Writing the Code for the GUI Callbacks](#)
- [Launching the GUI](#)
- [Troubleshooting and Potential Problems](#)
- [Related Posts and Other Links](#)

## Initializing GUIDE (GUI Creator)

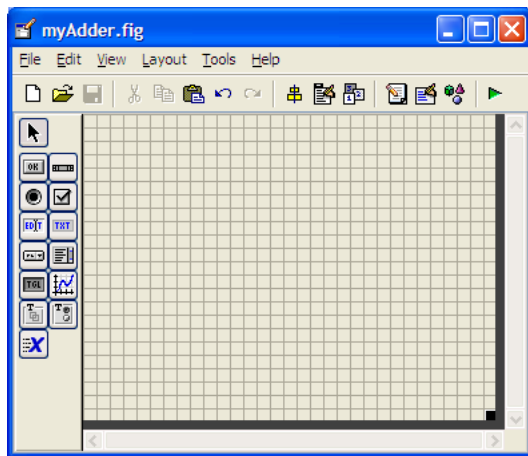
1. First, open up MATLAB. Go to the command window and type in guide.



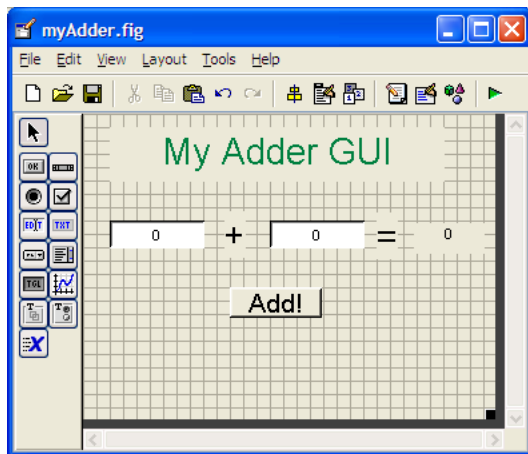
2. You should see the following screen appear. Choose the first option Blank GUI (Default).






3. You should now see the following screen (or something similar depending on what version of MATLAB you are using and what the predesignated settings are):



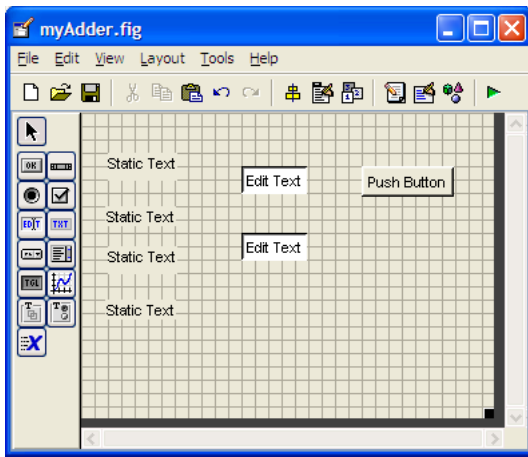
4. Before adding components blindly, it is good to have a rough idea about how you want the graphical part of the GUI to look like so that it'll be easier to lay it out. Below is a sample of what the finished GUI might look like.



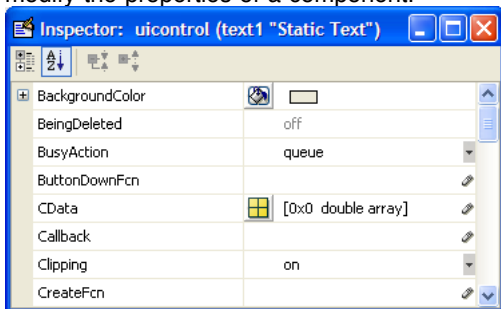
## Creating the Visual Aspect of the GUI: Part 1

1. For the adder GUI, we will need the following components
  -  Two Edit Text components
  -  Three Static Text component
  -  One Pushbutton component

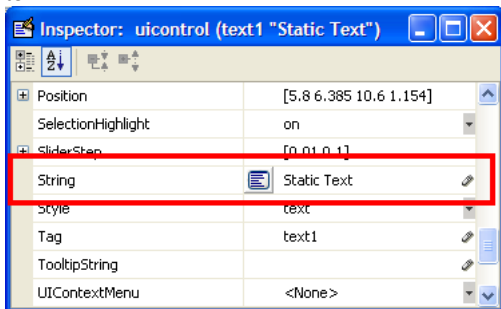
Add in all these components to the GUI by clicking on the icon and placing it onto the grid. At this point, your GUI should look similar to the figure below :



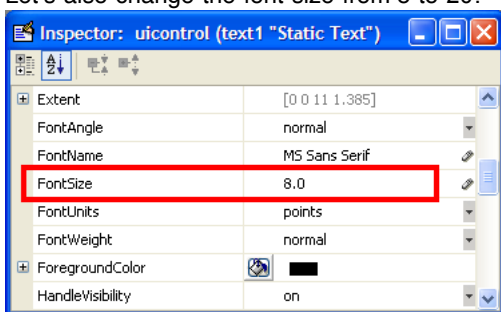
- Next, its time to edit the properties of these components. Let's start with the static text. Double click one of the *Static Text* components. You should see the following table appear. It is called the *Property Inspector* and allows you to modify the properties of a component.



- We're interested in changing the *String* parameter. Go ahead and edit this text to +.



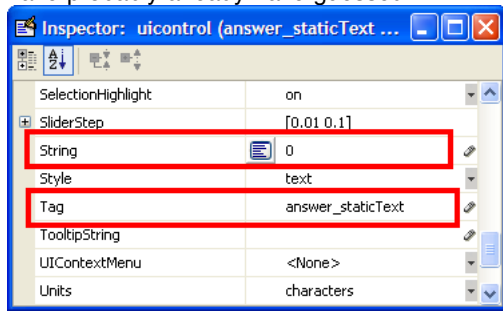
Let's also change the font size from 8 to 20.



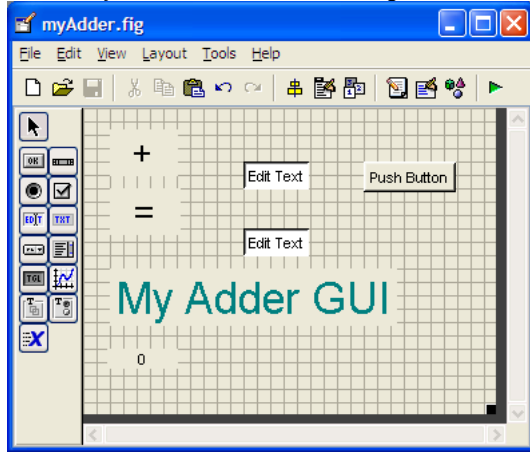
After modifying these properties, the component may not be fully visible on the GUI editor. This can be fixed if you resize the component, i.e. use your mouse cursor and stretch the component to make it larger.

- Now, do the same for the next *Static Text* component, but instead of changing the *String* parameter to +, change it to =.
- For the third *Static Text* component, change the *String* parameter to whatever you want as the title to your GUI. I kept it simple and named it MyAdderGUI. You can also experiment around with the different font options as well.
- For the final *Static Text* component, we want to set the *String* Parameter to 0. In addition, we want to modify the *Tag* parameter for this component. The *Tag* parameter is basically the variable name of this component. Let's call it answer\_staticText. This component will be used to display our answer, as you

have probably already have guessed.

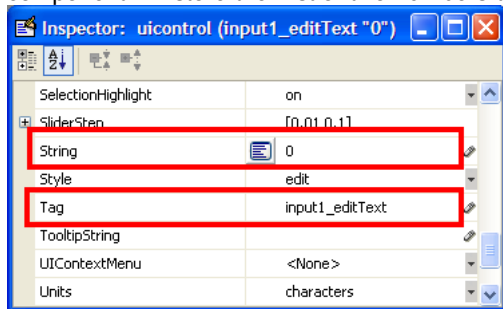


7. So now, you should have something that looks like the following:

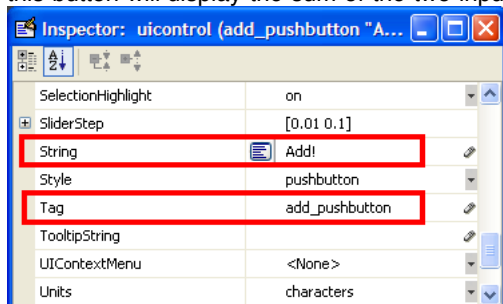


## Creating the Visual Aspect of the GUI: Part 2

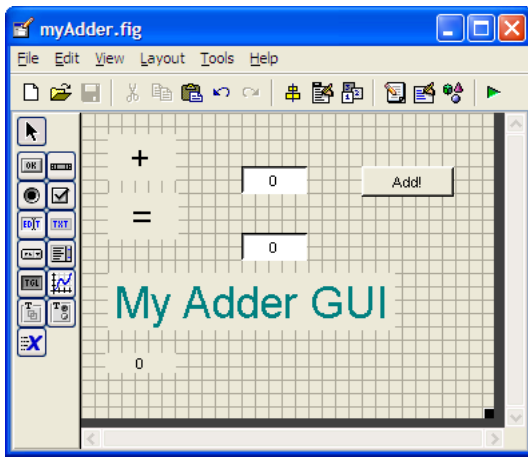
1. Next, lets modify the *Edit Text* components. Double click on the first *Edit Text* component. We want to set the *String* parameter to 0 and we also want to change the *Tag* parameter to input1\_editText, as shown below. This component will store the first of two numbers that will be added together.



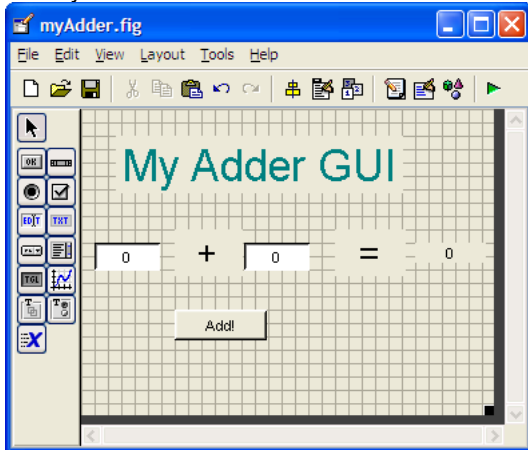
2. For the second *Edit Text* component, set the *String* parameter to 0 **BUT** set the *Tag* parameter input2\_editText. This component will store the second of two numbers that will be added together.
3. Finally, we need to modify the *pushbutton* component. Change the *String* parameter to Add! and change the *Tag* parameter to add\_pushbutton. Pushing this button will display the sum of the two input numbers.



4. So now, you should have something like this:




Rearrange your components accordingly. You should have something like this when you are done:

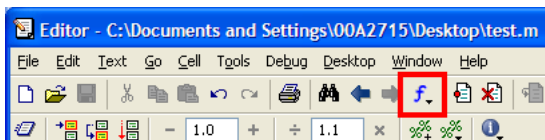


- Now, save your GUI under any file name you please. I chose to name mine *myAdder*. When you save this file, MATLAB automatically generates two files: *myAdder.fig* and *myAdder.m*. The *.fig* file contains the graphics of your interface. The *.m* file contains all the code for the GUI.

## Writing the Code for the GUI Callbacks

MATLAB automatically generates an *.m* file to go along with the figure that you just put together. The *.m* file is where we attach the appropriate code to the callback of each component. For the purposes of this tutorial, we are primarily concerned only with the *callback* functions. You don't have to worry about any of the other function types.

- Open up the *.m* file that was automatically generated when you saved your GUI. In the MATLAB editor, click on the  icon, which will bring up a list of the functions within the *.m* file. Select *input1\_editText\_Callback*.



- The cursor should take you to the following code block:

```
function input1_editText_Callback(hObject, eventdata, handles)
% hObject    handle to input1_editText (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,'String') returns contents of input1_editText as text
%       str2double(get(hObject,'String')) returns contents of
%       input1_editText as a double
```

Add the following code to the bottom of that code block:


```
%store the contents of input1_editText as a string. if the string
%is not a number then input will be empty
```

```
input = str2num(get(hObject,'String'));
```

```
%checks to see if input is empty. if so, default input1_editText to zero
if (isempty(input))
    set(hObject,'String','0')
end
guidata(hObject, handles);
```

This piece of code simply makes sure that the input is well defined. We don't want the user to put in inputs that aren't numbers! The last line of code tells the gui to update the handles structure after the callback is complete. The handles stores all the relevant data related to the GUI. This topic will be discussed in depth in a different tutorial. For now, you should take it at face value that it's a good idea to end each callback function with `guidata(hObject, handles)`; so that the handles are always updated after each callback. This can save you from potential headaches later on.

3. Add the same block of code to *input2\_editText\_Callback*.

4. Now we need to edit the *add\_pushbutton\_Callback*. Click on the  icon and select *add\_pushbutton\_Callback*. The following code block is what you should see in the .m file.

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

Here is the code that we will add to this callback:

```
a = get(handles.input1_editText,'String');
b = get(handles.input2_editText,'String');
% a and b are variables of Strings type, and need to be converted
% to variables of Number type before they can be added together

total = str2num(a) + str2num(b);
c = num2str(total);
% need to convert the answer back into String type to display it
set(handles.answer_staticText,'String',c);
guidata(hObject, handles);
```

5. Let's discuss how the code we just added works:

```
a = get(handles.input1_editText,'String');
b = get(handles.input2_editText,'String');
```

The two lines of code above take the strings within the *Edit Text* components, and stores them into the variables *a* and *b*. Since they are variables of *String* type, and not *Number* type, we cannot simply add them together. Thus, we must convert *a* and *b* to *Number* type before MATLAB can add them together.

6. We can convert variables of *String* type to *Number* type using the MATLAB command `str2num(String argument)`. Similarly, we can do the opposite using `num2str(Number argument)`. The following line of code is used to add the two inputs together.

```
total= (str2num(a) + str2num(b));
```

The next line of code converts the *sum* variable to *String* type and stores it into the variable *c*.

```
c = num2str(total);
```

The reason we convert the final answer back into *String* type is because the *Static Text* component does not display variables of *Number* type. If you did not convert it back into a *String* type, the GUI would run into an error when it tries to display the answer.

7. Now we just need to send the sum of the two inputs to the answer box that we created. This is done using the following line of code. This line of code populates the *Static Text* component with the variable *c*.

```
set(handles.answer_staticText,'String',c);
```

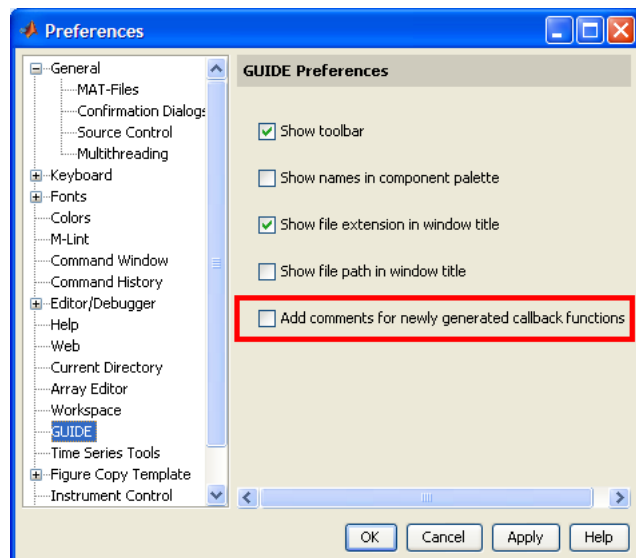
The last line of code updates the handles structures as was previously

mentioned.

```
guidata(hObject, handles);
```


Congratulations, we're finished coding the GUI. Don't forget to save your m-file. It is now time to launch the GUI!

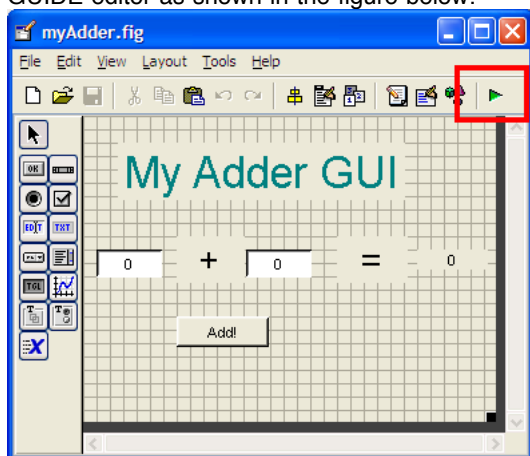
8. If you don't want MATLAB to automatically generate all those comments for each of the callbacks, there is a way to disable this feature. From the GUI editor, go to **File**, then to **Preferences**.



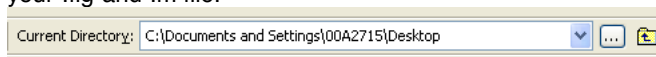
## Launching the GUI

1. There are two ways to launch your GUI.

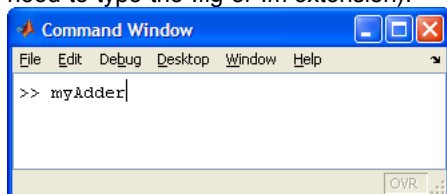
- The first way is through the GUIDE editor. Simply press the  icon on the GUIDE editor as shown in the figure below:



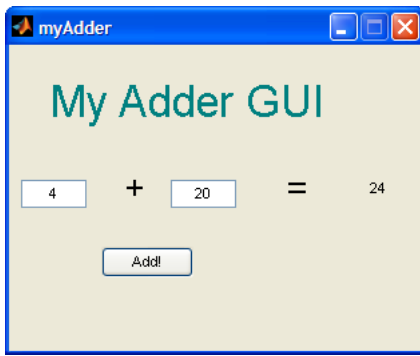
- The second method is to launch the GUI from the MATLAB command prompt. First, set the MATLAB current directory to wherever you saved your .fig and .m file.



Next, type in the name of the GUI at the command prompt (you don't need to type the .fig or .m extension):



2. The GUI should start running immediately:



Try to input some numbers to test out the GUI. **Congratulations** on creating your first GUI!

## Troubleshooting and Potential Problems

So your GUI doesn't work and you don't know why. Here are a couple of tips that might help you find your bug:

1. If you can't figure out where your error is, it might be a good idea to quickly go through this tutorial again.
2. The command line can give you many hints on where exactly the problem resides. If your GUI is not working for any reason, the error will be outputted to the command prompt. The line number of the faulty code and a short description of the error is given. This is always a good place to start investigating.
3. Make sure all your variable names are consistent in the code. In addition, make sure your component Tags are consistent between the .fig and the .m file. For example, if you're trying to extract the string from the *Edit Text* component, make sure that your get statement uses the right tag! More specifically, if you have the following line in your code, make sure that you named the *Edit Text* component accordingly!  

```
a = get(handles.input1_editText,'String');
```
4. The source code is available [here](#), and could be useful for debugging purposes.
5. If all else fails, leave a comment here and we'll try our best to help.



## Related Posts and Other Links

[MATLAB GUI Tutorial - Slider](#)  
[MATLAB GUI Tutorial - Pop-up Menu](#)  
[MATLAB GUI Tutorial - Plotting Data to Axes](#)  
[MATLAB GUI Tutorial - Button Types and Button Group](#)  
[MATLAB GUI Tutorial - A Brief Introduction to handles](#)  
[MATLAB GUI Tutorial - Sharing Data among Callbacks and Sub Functions](#)  
[Video Tutorial: GUIDE Basics](#)  
[More GUI Tutorial Videos From Doug Hull](#)

This is the end of the tutorial.



## 341 Responses to “MATLAB GUI (Graphical User Interface) Tutorial for Beginners”

1. on 20 Nov 2007 at 10:04 am [1Mike](#)

Thanks for the tutorial - its nice and clear 😊



# [blinkdagger](#)

an Engineering and MATLAB blog

- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

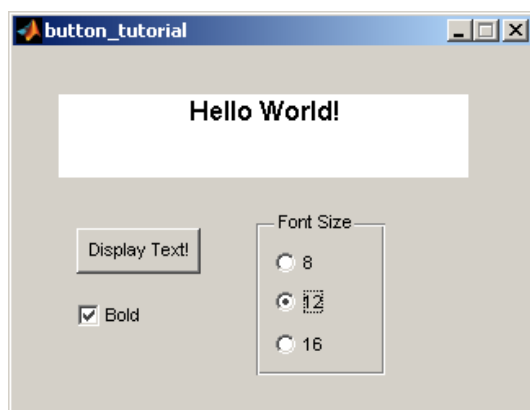
## [MATLAB GUI Tutorial - Button Types and Button Group](#)

03 Nov 2007 [Quan Quach](#) [93 comments](#) 25,002 views

### Introduction



In this three-part Matlab GUI Tutorial, you will learn how to use the different types of buttons available within Matlab GUIs. These button types are: push button, radio button, check box, and toggle buttons. In addition, you will learn how to use the button panel to control a group of buttons.

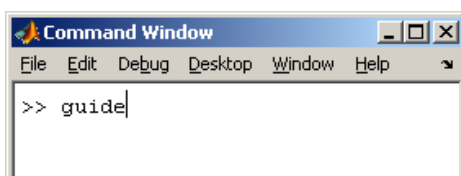


This tutorial is written for those with little or no experience creating a Matlab GUI (Graphical User Interface). If you're new to creating GUIs in Matlab, you should [visit this tutorial first](#). Basic knowledge of Matlab is recommended. Matlab version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Let's get started!

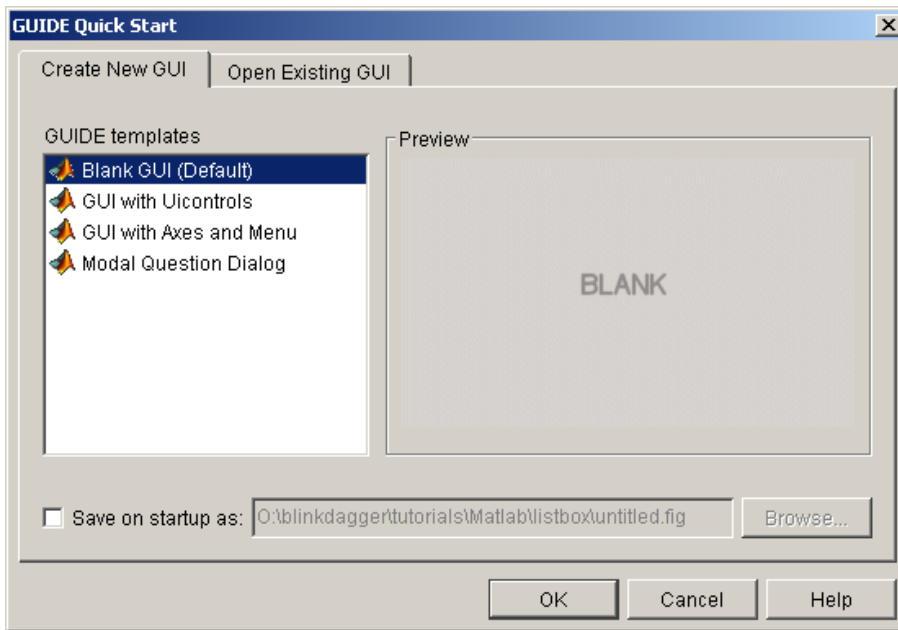
### Part One: The Pushbutton



The push button is a very simple component. When the user clicks on a push button, it causes an action to occur. This action is dictated by the code that is programmed in the push button's callback function. In this part of the tutorial, we will program the push button to display some text when it is pressed.

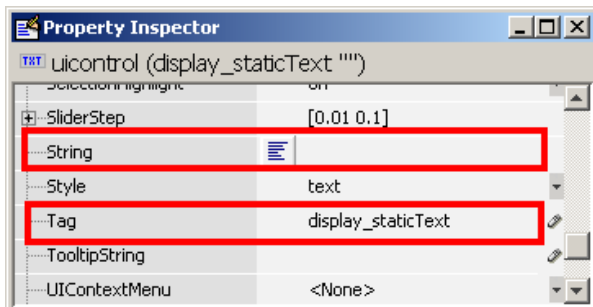
1. First, we are going to create the visual aspect of the GUI. Open up Matlab. Go to the command window and type in guide.



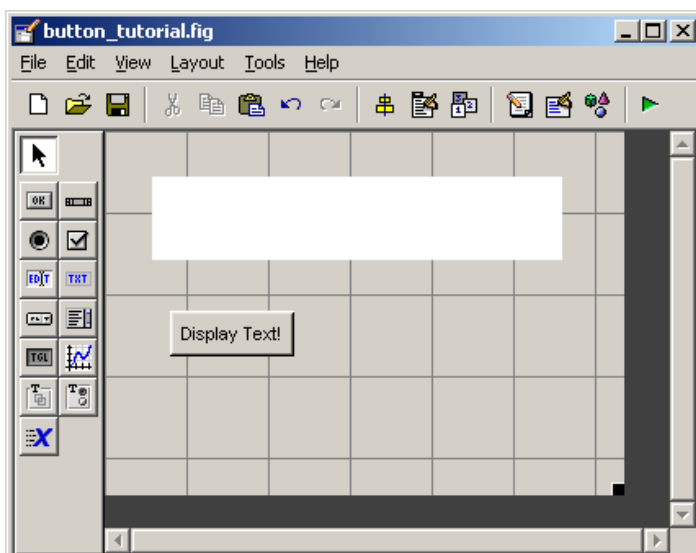
- You should see the following screen appear. Choose the first option Blank GUI (Default).




- Click on  and add one *Static Text* component to the GUI figure. Next, click on  and add one *Push button* component onto the GUI figure.
- Double click the *Static Text* component to bring up the Property Inspector. Change the *String* property so that there is nothing inside. Change the *Tag* property to `display_staticText`. Similarly, double click on the *Pushbutton* component and change the *String* property to `Display Text!` and change the *Tag* property to `displayText_pushbutton`.

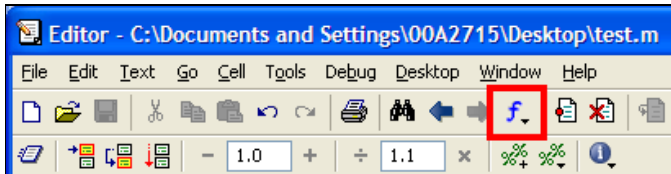


- Here's what your figure should look like after you add the components and modify them.





6. Save your GUI wherever you please with your desired filename.
7. Now, we are going to write the code for the GUI. When you save your GUI, Matlab automatically generates an .m file to go along with the figure that you just put together. The .m file is where we attach the appropriate code to the callback of each component. For the purposes of this tutorial, we are primarily concerned only with the callback functions. You don't have to worry about any of the other function types.

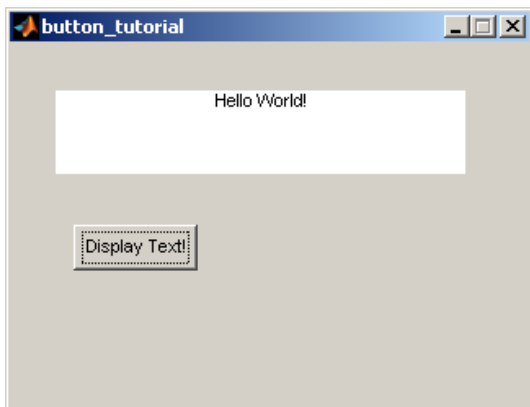
Open up the .m file that was automatically generated when you saved your GUI. In the Matlab editor, click on the  icon, which will bring up a list of the functions within the .m file. Select `displayText_pushbutton_Callback`.



Add the following code to the function:

```
%display "Hello Word!!" in the static text component when the  
%pushbutton is pressed  
set(handles.display_staticText, 'String', 'Hello World!');
```


8. Now that we've completed both the visual and code aspects of the GUI, its time to run the GUI to make sure it works before we move on. From the m-file editor, you can click on the  icon to save and run the GUI. Alternatively, from the GUIDE editor, you can click on the  to launch the GUI. The GUI should appear once you click the icon. Now try clicking on the button to make sure that Hello World! appears on the GUI.



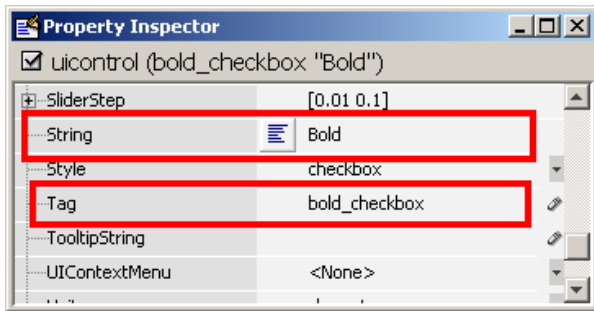
And that's it. Those are the basics of using the *Push button* component. Now we're ready to move onto the *Check box* component.

## Part Two: The Check Box

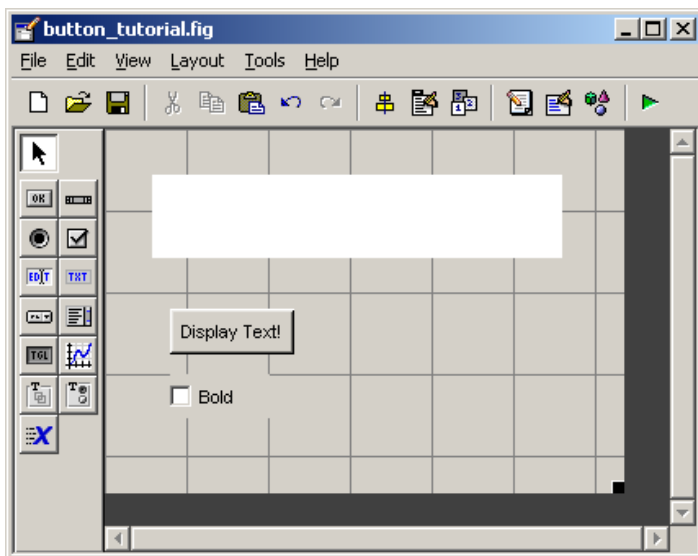
The *Check Box* component has two states, *checked* and *unchecked*. These components are usually used to control options that can be turned *on* and *off*. In this part of the tutorial, we will add the functionality of making the display text become bold or unbolded.

1. The first thing we need to do is to add a *Check Box* Component to the GUI figure that we were just working with. So if you closed GUIDE, reopen it again. Once you have GUIDE opened again, Click on  and add one *Check Box* component to the GUI figure.

2. Double click the *Check Box* component to bring up the Property Inspector. Change the *String* property to Bold. Change the *Tag* property to bold\_checkbox.



3. Here's what your figure should look like after you add the *Check Box* component and modify it.

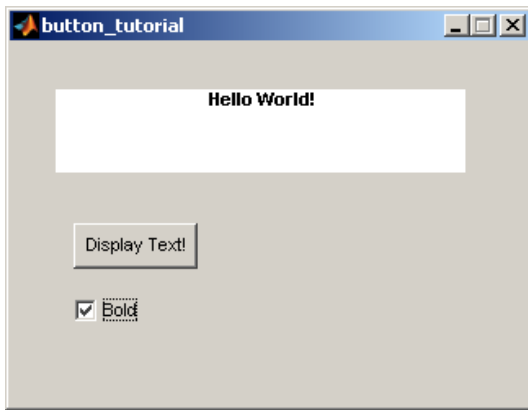


4. Add the following code to the *bold\_checkbox\_Callback* function:

```
%checkboxStatus = 0, if the box is unchecked,  
%checkboxStatus = 1, if the box is checked  
checkboxStatus = get(handles.bold_checkbox, 'Value');  
if(checkboxStatus)  
    %if box is checked, text is set to bold  
    set(handles.display_staticText, 'FontWeight', 'bold');  
else  
    %if box is unchecked, text is set to normal  
    set(handles.display_staticText, 'FontWeight', 'normal');  
end
```

**Note:** The *bold\_checkbox\_Callback* function triggers when the user activates the check box **AND** when the user deactivates the check box.

5. Now that we've completed both the visual and code aspects of the GUI, its time to run the GUI to make sure it works before we move on. Try checking and unchecking the *Check Box* component to make sure that the text "Hello World!" is being bolded and unbolded.





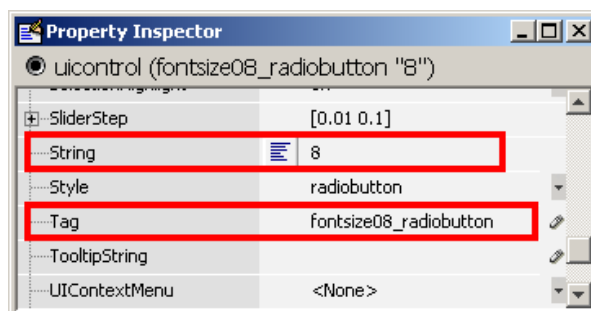
And that's it. Those are the basics of using the *Check Box* component. Now we're ready to move onto the *Button Group*, which is the most challenging part of this tutorial.

## Part Three: Radio Buttons, Toggle Buttons, and Button Group Panel

Radio buttons and Toggle buttons are used exactly the same way that check boxes are used in Matlab GUIs, so we won't go over how to use them. But there is one special case that needs to be covered. When either radio buttons or toggle buttons are used in conjunction with the button group panel, they exhibit mutually exclusive behavior. Simply put, this means that only one radio button or one toggle button can be selected at a time. This behavior can come in very useful for some GUIs. Since radio buttons and toggle buttons are identical in their functionality, what is said about one, is true for the other. Thus, only radio buttons will be discussed from here on out.

In this part of the tutorial, we will create a button group that will allow you to choose between different font sizes for the display text.

1. The first thing we need to do is to add a *Button Panel* component to the GUI figure that we were just working with. So if you closed GUIDE, reopen it again. Once you have GUIDE opened again, click on  and add one *Button Panel* component to the GUI figure. Make sure it's large enough to fit in three radio buttons. Next, click on  and add three radio buttons onto the button group panel.
2. Double click on the first *Radio Button* component to bring up the Property Inspector. Change the *String* property to 8. Change the *Tag* property to fontsize08\_radiobutton.

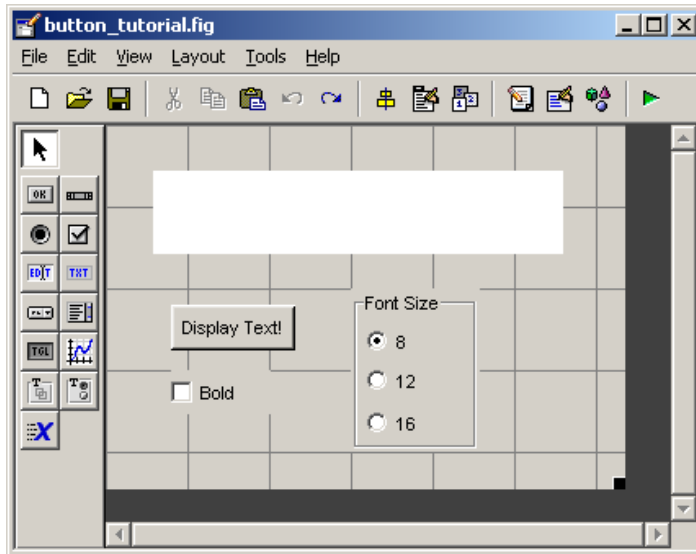



Next, double click on the second *Radio Button* component, and change the *String* property to 12, and change the *Tag* property to fontsize12\_radiobutton.

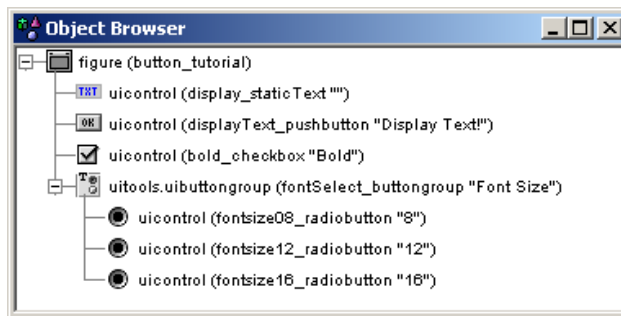
Next, double click on the third *Radio Button* component, and change the *String* property to 16, and change the *Tag* property to fontsize16\_radiobutton.

Finally, double click on the button group panel and change the *Tag* property to fontSelect\_buttongroup. You should also change the *String* property for the button group panel to Fontsize.

- Here's what your figure should look like after you add the components and modify them.



- Before we move on, we should check the hierarchical structure of the GUI figure. Click on the  icon and the following should appear:



Make sure that the three radio buttons are one hierarchy below the button group icon.

- Add the following line of code to the opening function. In this tutorial example, it is named *button\_tutorial\_OpeningFcn* function. Yours will be the name of the file you saved it as, followed by "\_OpeningFcn".

```
set(handles.fontSelect_buttongroup,'SelectionChangeFcn',@fontSelect_buttongroup_SelectionChangeFcn);
```

Make sure the previous line was added right before the line:

```
guidata(hObject, handles);
```

Next, add the following function at the very end of the .m file.

```
function fontSelect_buttongroup_SelectionChangeFcn(hObject, eventdata)
```

```
%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);
```

```
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object
case 'fontsize08_radiobutton'
    %execute this code when fontsize08_radiobutton is selected
    set(handles.display_staticText,'FontSize',8);
```

```

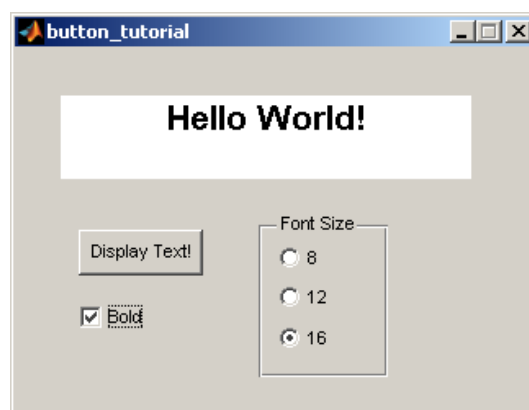
case 'fontsize12_radiobutton'
%execute this code when fontsize12_radiobutton is selected
set(handles.display_staticText,'FontSize',12);

case 'fontsize16_radiobutton'
%execute this code when fontsize16_radiobutton is selected
set(handles.display_staticText,'FontSize',16);
otherwise
% Code for when there is no match.

end
%updates the handles structure
guidata(hObject, handles);

```

6. Notice that the callback functions for the radio buttons were not automatically generated by Matlab. This is completely normal. Each time a button is selected within the *Button Group Panel* component, the function defined within the *SelectionChangeFcn* property of *Button Group Panel* component is called. The line of code that was added in the opening function specifies the callback function when a button within the button group is selected. The selection change function is then defined at the end of the .m file.
7. Now that we've completed both the visual and code aspects of the GUI, its time to run the GUI again. Try clicking on all of the buttons to make sure they perform their function correctly. Specifically, make sure that the font size changes accordingly.



And that's it. Those are the basics of using the different buttons within the Matlab GUI.

## Download Source Files

Source files can be downloaded [here](#).

This is the end of the tutorial.



## 93 Responses to “MATLAB GUI Tutorial - Button Types and Button Group”

1. on 14 Dec 2007 at 1:36 am [1](#) *Tongtong*

A very good tutorial for the beginners. Thanks.

2. on 14 Dec 2007 at 2:09 am [2](#) *Tongtong*

# blinkdagger

an Engineering and MATLAB blog

- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

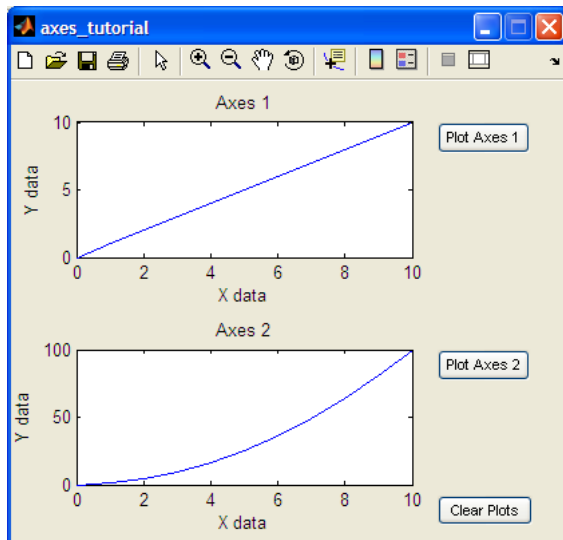
## MATLAB GUI Tutorial - Plotting Data to Axes

31 Oct 2007 [Quan Quach](#) 176 comments 25,632 views

### Introduction



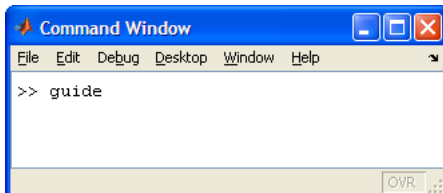
In this Matlab GUI tutorial, you will learn how to create and use the *Axes* component. The *Axes* component allows you to display graphics, such as graphs and images on your GUI. In this tutorial, we will create two axes on the GUI and plot some simple data onto it. In addition, we will include a reset button to clear the axes and we will also add the standard toolbar to allow the user to zoom, pan, and query the plot.



This tutorial is written for those with little or no experience creating a Matlab GUI (Graphical User Interface). If you're new to creating GUIs in Matlab, you should [visit this tutorial first](#). Basic knowledge of Matlab is recommended. Matlab version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Let's get started!

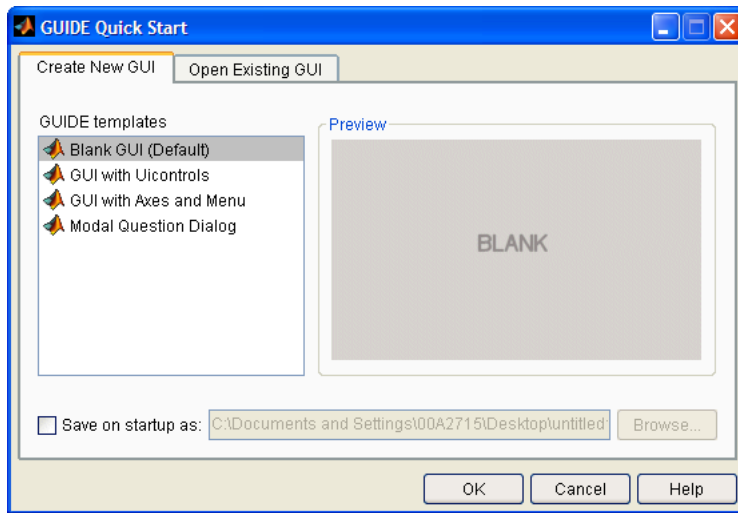
### Create the Visual Aspect of the GUI



1. First, open up Matlab. Go to the command window and type in guide.

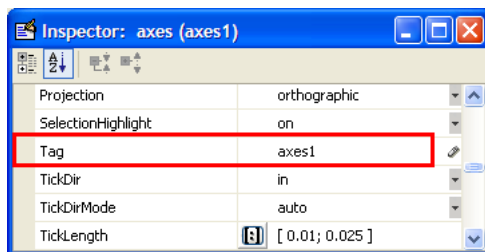


2. You should see the following screen appear. Choose the first option Blank GUI (Default).

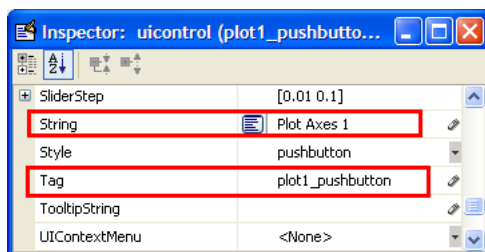




3. Click on  and add two *Axes* components to the GUI figure. Next, click on  and add three *Pushbutton* components onto the GUI figure.
4. Double click the *Axes* component to bring up the Property Inspector. Change the *Tag* property to *axes1*, which should already be the default name. Additionally, make sure the other *Axes* component's *Tag* property is named *axes2*.



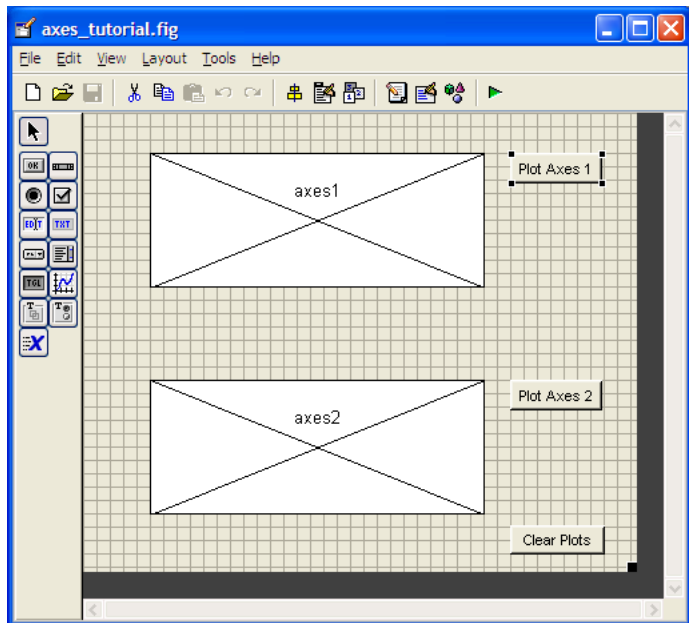
5. Next, let's modify the properties of the *Pushbutton* components. Double click on one of the *Pushbutton* components. Change the *String* property to Plot Axes 1, and the *Tag* property to *plotAxes1\_pushbutton*, as shown below.



Similarly, double click on the next pushbutton and change the *String* property to Plot Axes 2 and change the *Tag* property to *plotAxes2\_pushbutton*.

Finally, double click on the final pushbutton and change the *String* property to Clear Axes and change the *Tag* property to *clearAxes\_pushbutton*.


6. Here's what your figure should look like after you add the components and modify them.

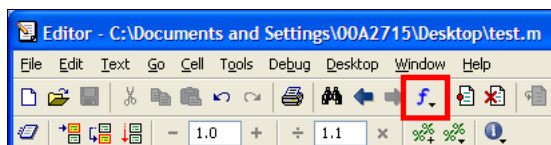


7. Save your GUI wherever you please with your desired filename.

## Writing the Code for the GUI

Matlab automatically generates an .m file to go along with the figure that you just put together. The .m file is where we attach the appropriate code to the callback of each component. For the purposes of this tutorial, we are primarily concerned only with the callback functions. You don't have to worry about any of the other function types.

1. Open up the .m file that was automatically generated when you saved your GUI. In the Matlab editor, click on the  icon, which will bring up a list of the functions within the .m file. Select *plot1\_pushbutton\_Callback*.



Add the following code to the function:

```
%selects axes1 as the current axes, so that
%Matlab knows where to plot the data
axes(handles.axes1)

%creates a vector from 0 to 10, [0 1 2 3 . . . 10]
x = 0:10;
%creates a vector from 0 to 10, [0 1 2 3 . . . 10]
y = 0:10;

%plots the x and y data
plot(x,y);
%adds a title, x-axis description, and y-axis description
title('Axes 1');
xlabel('X data');
ylabel('Y data');
guidata(hObject, handles); %updates the handles
```

2. Similarly, we want to put the following code into the *plot2\_pushbutton\_Callback*:

```
%selects axes2 as the current axes, so that
%Matlab knows where to plot the data
axes(handles.axes2)

%creates a vector from 0 to 10, [0 1 2 3 . . . 10]
x = 0:10;
%creates a vector [0 1 4 9 . . . 100]
y = x.^2
```

```
%plots the x and y data
plot(x,y);
%adds a title, x-axis description, and y-axis description
title('Axes 2');
xlabel('X data');
ylabel('Y data');
guidata(hObject, handles); %updates the handles
```

- Next, we need to add some code to the *clearPlots\_pushbutton\_Callback*:

```
%these two lines of code clears both axes
cla(handles.axes1,'reset')
cla(handles.axes2,'reset')
guidata(hObject, handles); %updates the handles
```

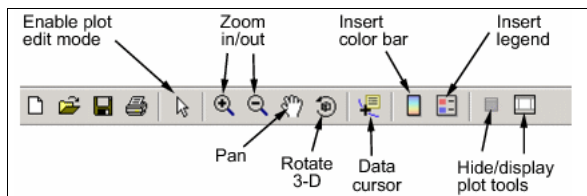
- And finally, we need to add the following line of code to *axes\_tutorial\_OpeningFcn*:

```
set(hObject,'toolbar','figure');
```

This line of code should be placed right before:

```
guidata(hObject, handles);
```



This line of code effectively adds the standard toolbar to the GUI, allowing the user to zoom, pan, query the plot, and more. The standard toolbar and a brief description of the icons are shown below:

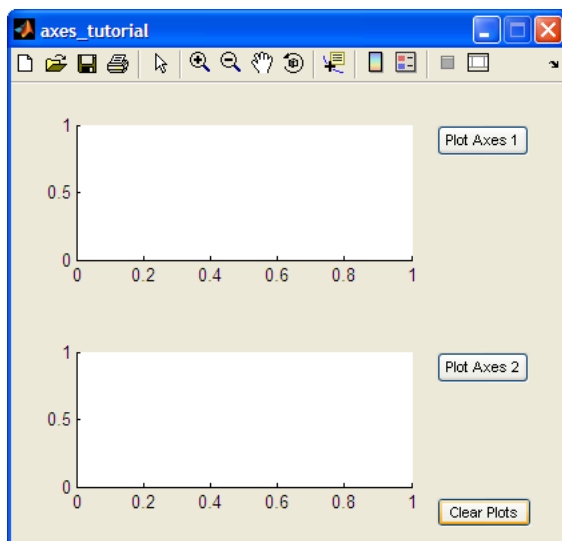


- Save your m-file!

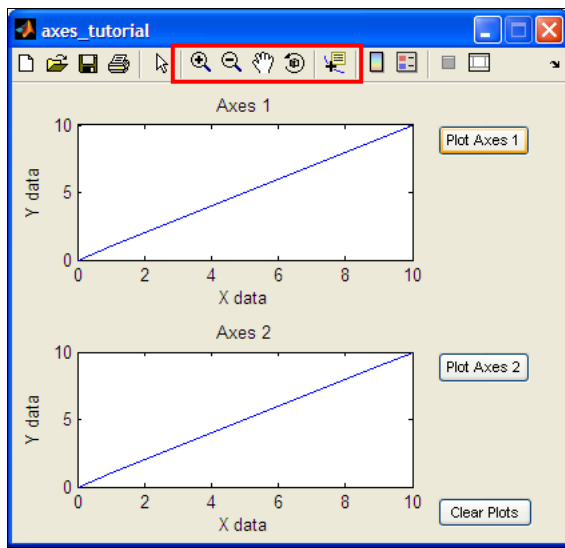
## Run and Test the GUI

Now that we've completed both the visual and code aspects of the GUI, its time to run the GUI to make sure it works.

- From the m-file editor, you can click on the  icon to save and run the GUI. Alternatively, from the GUIDE editor, you can click on the  to launch the GUI. The following GUI should appear once you click the icon:



- Go ahead and try pressing all of the buttons to make sure they work. If everything was done correctly, you should see the following plots. Also, you can use the icons that are within the red box to test out the other functions.



3. And that's it. Those are the basics of using the Axes component. You can explore the other options that the axes has to offer through the Property Inspector.

This is the end of the tutorial.

Source files can be downloaded [here](#).



## 176 Responses to “MATLAB GUI Tutorial - Plotting Data to Axes”

1. on 12 Jan 2008 at 12:48 pm [1Vaibhav Bedia](#)

When i close MATLAB and start guide again my axes loses its TAG and hence becomes invisible.How do i solve this?

2. on 20 Jan 2008 at 4:48 pm [2Alex](#)

nice one! U need to write some more stuff for beginners 😊

3. on 13 Feb 2008 at 3:59 pm [3Saikat](#)

Thank you so much ..... I love your style and find your instructions very helpful. Going through the programs you have written for us, the beginners, I feel more confident of writing my own codes .....  
Thank you once again .....

4. on 17 Feb 2008 at 11:03 am [4Tanty](#)

What a great tutorial.  
Im working on my final project, so this tutorial help me solve my problem writing the code for axes.  
Thanks.

5. on 17 Mar 2008 at 9:57 pm [5lovejoy](#)

how to plot a resultant vector in GUI ex.10N+5N

6. on 26 Mar 2008 at 4:28 am [6nola](#)

very nice tutorial..however i had a query for you..  
when i do a zoom on the image..and click one pushbutton to make an action..the image return to its initial dimensions(i loose the zoom)...could you help me on this subject?

7. on 26 Mar 2008 at 12:37 pm [7Daniel Sutoyo](#)

Hi Nola

good question... although I haven't tried to code it yet, what you need to do is to get the new axes info and store it somewhere.

For example, your axes is called handles.axes1. You will have to use the get() and get the min,max axis on handles.axes1 and store in a variable for example myaxis.

# [blinkdagger](#)

an Engineering and MATLAB blog

- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

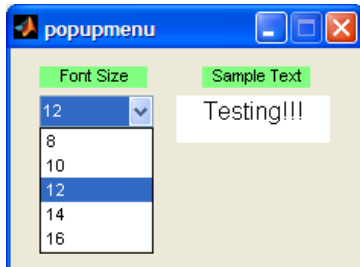
## [Matlab GUI Tutorial - Pop-up Menu](#)

29 Oct 2007 [Quan Quach](#) [137 comments](#) 20,912 views

### Introduction



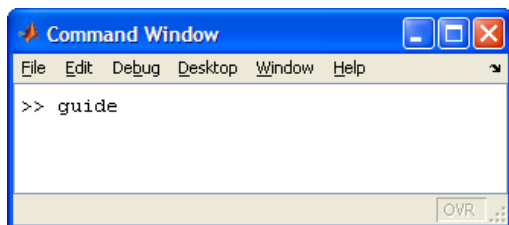
In this Matlab GUI tutorial, you will learn how to create and use the *Pop-up Menu* component. Pop-up menus are used as a control for choosing between a set of options. When the user clicks on the Pop-up menu, the menu expands, revealing a set of choices that the user can pick. A common use for Pop-up menus is a font size selector (shown below).



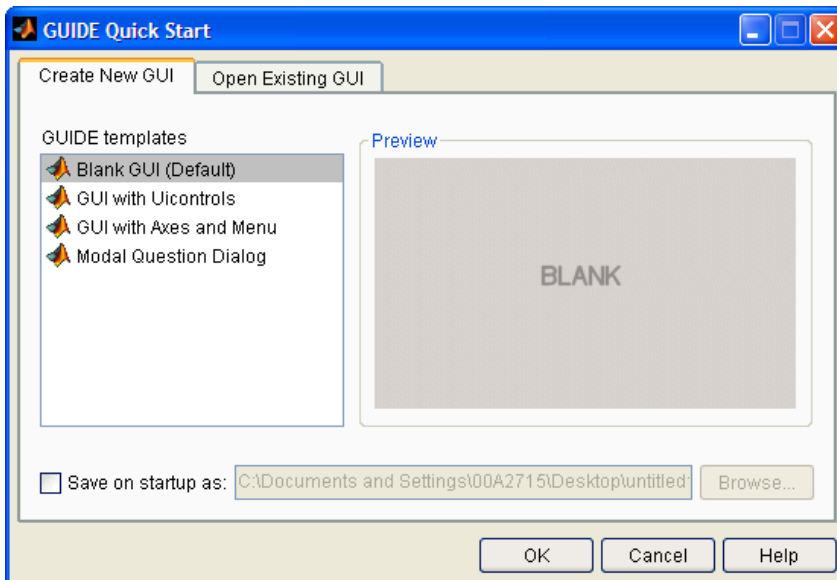
This tutorial is written for those with little or no experience creating a Matlab GUI (Graphical User Interface). If you're new to creating GUIs in Matlab, you should [visit this tutorial first](#). Basic knowledge of Matlab is not required, but recommended. Matlab version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Let's get started!



### Create the Visual Aspect of the GUI

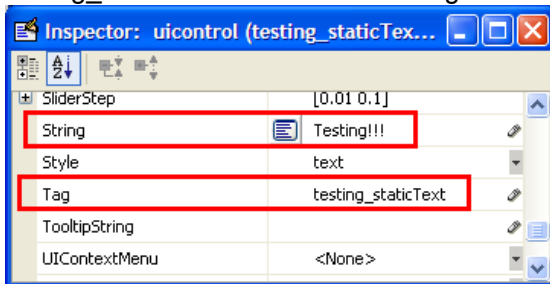
1. First, open up Matlab. Go to the command window and type in guide.



2. You should see the following screen appear. Choose the first option Blank GUI (Default).

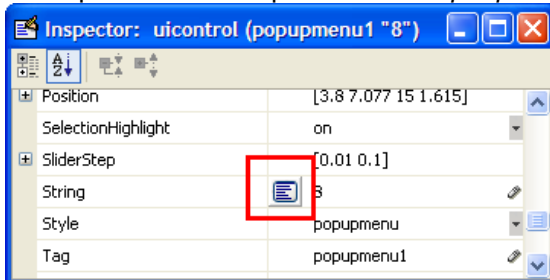


- Click on  and add a *Static Text* component to the GUI figure. Next, click on  and add a *Pop-up Menu* component onto the GUI figure.
- Double click the *Static Text* component to bring up the Property Inspector. Change the *String* property to Testing!!!, and change the *Tag* property to testing\_staticText as shown in the figure below:

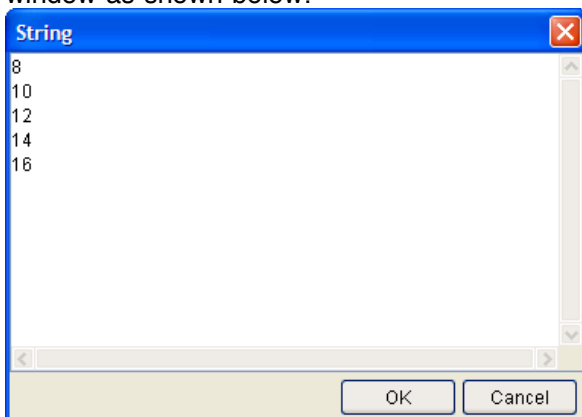


You can also modify the *BackgroundColor* property if you desire.

- Next, let's modify the properties of the *Pop-up Menu* component. First, click on the icon on the *String* property line as shown below. This allows you to edit the description for each option in the *Pop-Up Menu*.

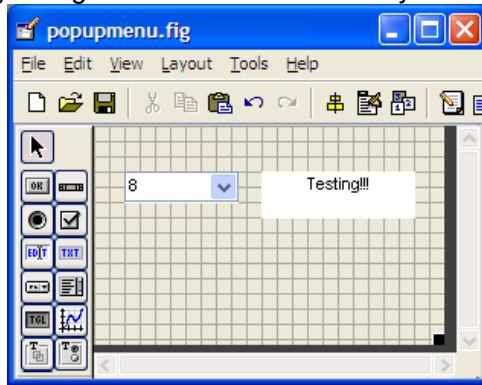


After clicking on the icon, you should now see the following window. Fill in the window as shown below:



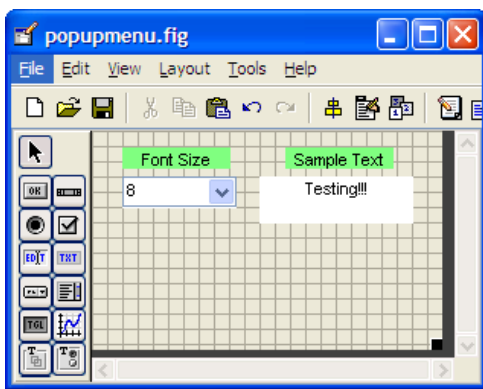
In addition, I set the *Tag* property to `popupmenu1`, which is the default name. You might want to make sure that its named properly before you move on.

- Here's what your figure should look like after you add the components and



modify them.


- At this point, you also might want to add some *Static Text* components to add some description tags to the GUI. You can modify their text by double clicking on the component and changing the *String* property. It's not required, but I highly recommend it.

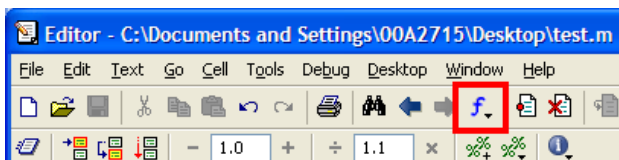


- Save your GUI wherever you please with your desired filename.

## Writing the Code for the GUI

Matlab automatically generates an `.m` file to go along with the figure that you just put together. The `.m` file is where we attach the appropriate code to the callback of each component. For the purposes of this tutorial, we are primarily concerned only with the callback functions. You don't have to worry about any of the other function types.

- Open up the `.m` file that was automatically generated when you saved your GUI. In the Matlab editor, click on the  icon, which will bring up a list of the functions within the `.m` file. Select `popupmenu1_Callback`.



Add the following code to the function:

```
%gets the selected option
switch get(handles.popupmenu1,'Value')
    case 1
        set(handles.testing_staticText,'FontSize',8);
    case 2
        set(handles.testing_staticText,'FontSize',10);
    case 3
        set(handles.testing_staticText,'FontSize',12);
```

```

case 4
    set(handles.testing_staticText,'FontSize',14);
case 5
    set(handles.testing_staticText,'FontSize',16);
otherwise
end

```

2. Lets quickly go over the code now. The following line of code gets the option that the user selected. Remember that in the visual layout of the GUI, we designated five different font sizes for the *Pop-up Menu* component, giving us five different options for the *Pop-up Menu*. So for example, if the user selected a font size of 8 (which was the first option in the *Pop-up Menu*), then the following line of code would return a value of 1. If the user selected a font size of 10, then the value returned would be 2, and so on.

```
get(handles.popupmenu1,'Value')
```

Depending on the option selected, the font of the *Static Text* component will be adjusted using the following line of code for each case statement:

```



%where ## is the appropriate fontsize value
set(handles.testing_staticText,'FontSize',##);

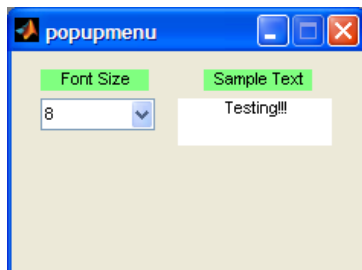
```

3. Save your m-file!

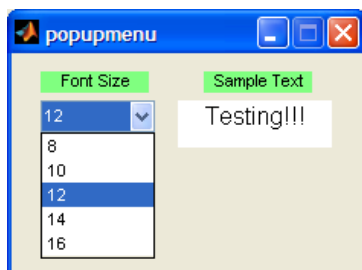
## Run and Test the GUI

Now that we've completed both the visual and code aspects of the GUI, its time to run the GUI to make sure it works.

1. From the m-file editor, you can click on the  icon to save and run the GUI. Alternatively, from the GUIDE editor, you can click on the  to launch the GUI. The following GUI should appear once you click the icon:



2. Go ahead and try selecting different font sizes. If everything was done correctly, you should see the font size of the sample text change accordingly.



3. And that's it. Those are the basics of using a *Pop-up Menu* component. You can explore the other options that the slider has to offer through the Property Inspector.

This is the end of the tutorial.

Source files can be downloaded [here](#).



# blinkdagger

an Engineering and MATLAB blog

- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

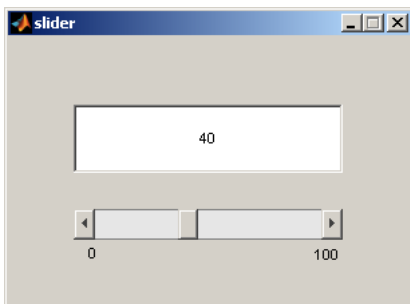
## MATLAB GUI Tutorial - Slider

28 Oct 2007 [Quan Quach](#) 118 comments 20,071 views

### Introduction



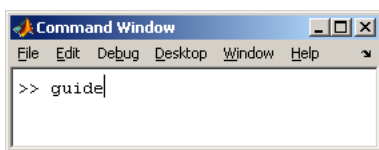
In this Matlab GUI tutorial, you will learn how to create and use the slider component. Sliders are useful controls for choosing a value in a range of values. Common uses are volume controls, seekers for movie and sound files as well as color pickers. An example of a slider is shown below.



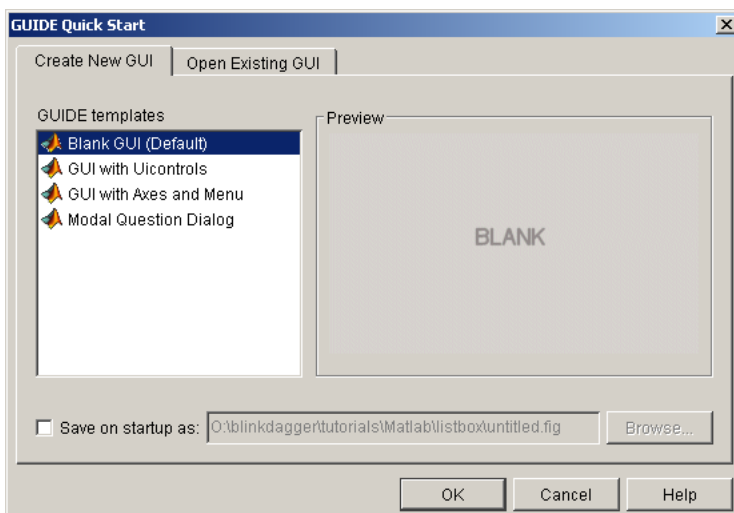
This tutorial is written for those with little or no experience creating a Matlab GUI (Graphical User Interface). If you're new to creating GUIs in Matlab, you should [visit this tutorial first](#). Basic knowledge of Matlab is not required, but recommended. Matlab version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Let's get started!



### Create the Visual Aspect of the GUI

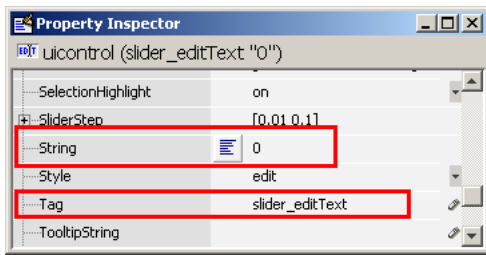
1. Start up Matlab, and type guide in the command line.



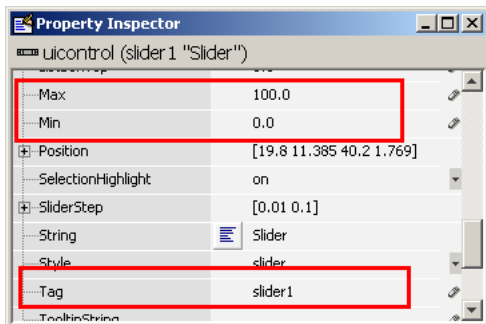
2. Choose to create a new GUI using the "Blank GUI(Default)" option.



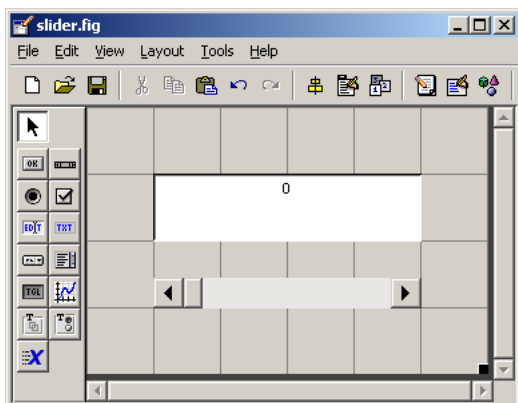
- Click on  and add an *Edit Text* component to the GUI figure. Next, click on  and add a *Slider* component onto the GUI figure.
- Double click the *Edit Text* component to bring up the Property Inspector. Change the *String* property to 0, and change the *Tag* property to sliderValue\_editText as shown in the figure below:



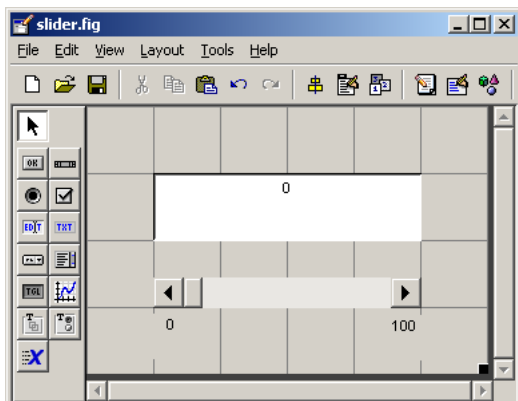
- Next, let's modify the properties of the *Slider* component. First let's set the *Min* property to 0, and the *Max* property to 100. Next, change the *Tag* property to slider1.



- Here's what your figure should look like after you add the components and modify them.




- At this point, you also might want to add some *Static Text* components to specify the min and max values of the slider. You can modify their text by double clicking on the component and changing the *String* property. It's not required, but I highly recommend it.

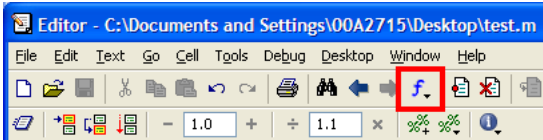


- Save your GUI wherever you please with your desired filename.

## Writing the Code for the GUI

Matlab automatically generates an .m file to go along with the figure that you just put together. The .m file is where we attach the appropriate code to the callback of each component. For the purposes of this tutorial, we are primarily concerned only with the callback functions. You don't have to worry about any of the other function types.

1. Open up the .m file that was automatically generated when you saved your GUI. In the Matlab editor, click on the  icon, which will bring up a list of the functions within the .m file. Select *slider1\_Callback*.



Add the following code to the function:

```
%obtains the slider value from the slider component
sliderValue = get(handles.slider1,'Value');

%puts the slider value into the edit text component
set(handles.slider_editText,'String', num2str(sliderValue));

% Update handles structure
guidata(hObject, handles);
```

2. Now, let's add the following code to the *slider\_editText\_Callback* function:

```
%get the string for the editText component
sliderValue = get(handles.slider_editText,'String');



%convert from string to number if possible, otherwise returns empty
sliderValue = str2num(sliderValue);

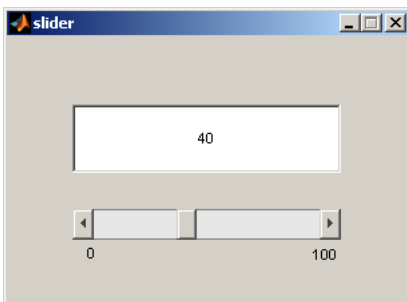
%if user inputs something is not a number, or if the input is less than 0
%or greater than 100, then the slider value defaults to 0
if (isempty(sliderValue) || sliderValue < 0 || sliderValue > 100)
    set(handles.slider1,'Value',0);
    set(handles.slider_editText,'String','0');
else
    set(handles.slider1,'Value',sliderValue);
end
```

3. Save your m-file!

## Run and Test the GUI

Now that we've completed both the visual and code aspects of the GUI, its time to run the GUI to make sure it works.

1. From the m-file editor, you can click on the  icon to save and run the GUI. Alternatively, from the GUIDE editor, you can click on the  to launch the GUI. The following GUI should appear once you click the icon:



2. Now, try to put in different types of inputs to test the GUI. Any input that is not a number, less than zero, or greater than 100 should default the slider to a value of zero.
3. And that's it. Those are the basics of using a *Slider* component. You can

explore the other options that the slider has to offer through the Property Inspector. For instance, you can use the *SliderStep* property to customize how far you want the slider to move when you press the left and right arrow, or when you click on the scroll bar.

This is the end of the tutorial.

Source files can be downloaded [here](#).



# [blinkdagger](#)

an Engineering and MATLAB blog

- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

## [MATLAB GUI Tutorial - Sharing Data among Callbacks and Sub Functions](#)

14 Nov 2007 [Quan Quach](#) [69 comments](#) 14,141 views

### Introduction



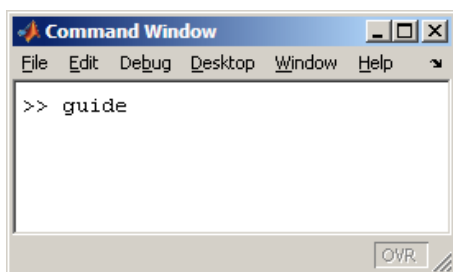
For the majority of GUIs that you create in Matlab, you will have to be able to share data among the component callbacks. One way of accomplishing this is to use the handles structure to store that data. In this tutorial, you will learn how to do so.

This tutorial is written for those with little experience creating a Matlab GUI (Graphical User Interface). If you're new to creating GUIs in Matlab, you should [visit this tutorial first](#). Basic knowledge of Matlab is recommended. Matlab version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Let's get started!

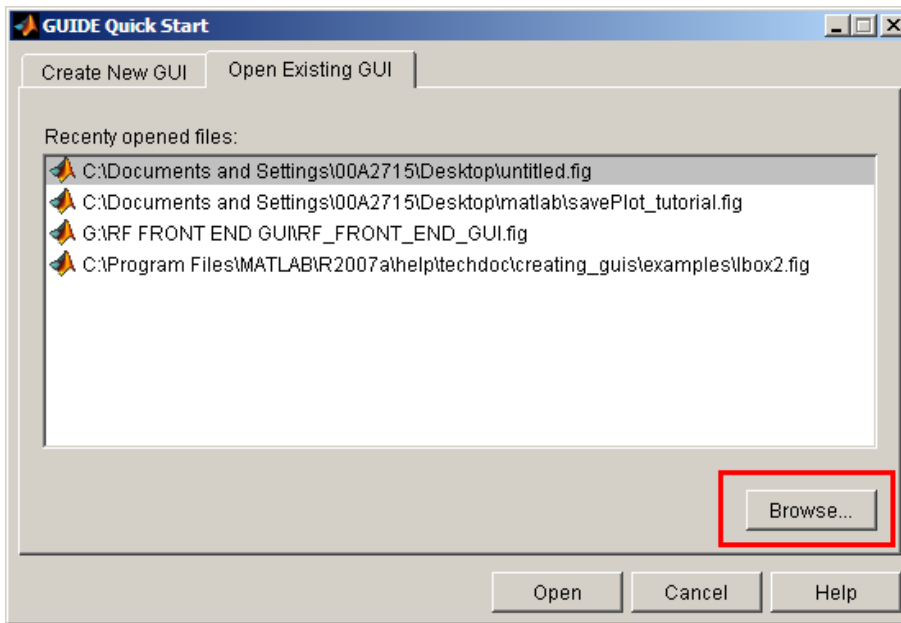
### Sharing Data Between Callbacks

A simple example of sharing data among callbacks is to increment a numerical counter on the GUI each time ANY button on the GUI is pressed. How can this be done? This tutorial will explain how.

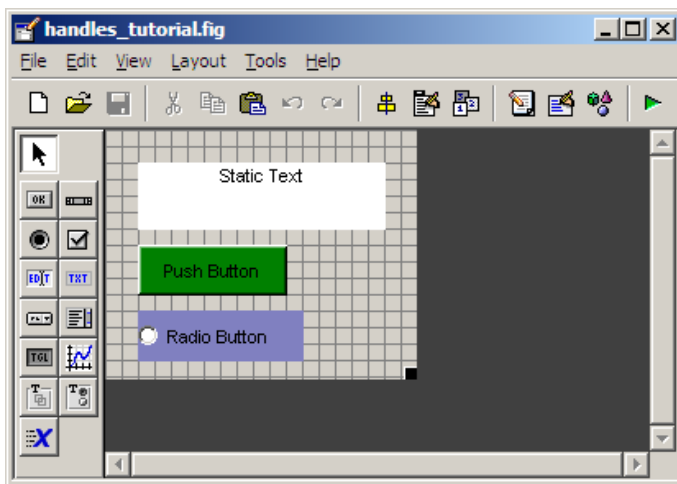
1. First, download the sample GUI [here](#). Unzip the files and place them wherever you please.
2. Now, type guide at the command prompt.




3. Choose to open the sample GUI by clicking on "Open Existing GUI". Click on "Browse" to locate where you saved the GUI files.



4. Here is what the GUI should look like when you open it:



5. Click on the  icon on the GUI figure to bring up the accompanying .m file.
6. The first thing we need to do is to define a variable that will hold the value for our numerical counter. Place the following code in *handles\_tutorial\_OpeningFcn*.

```
handles.buttonCounter = 0;
set(handles.text1,'String','0');
```

Make sure you place it right before this line of code:

```
guidata(hObject, handles);
```

7. Notice that we defined the variable name as `handles.buttonCounter`, rather than `buttonCounter`. If we did not define the variable within the `handles` structure, then it would be considered a local variable. Being a local variable means that the variable does not exist outside of the function where it was defined. Thus we need to store it into the `handles` structure, so that it can be accessed and modified later in the other callbacks. So, if you want data to be available to ALL callbacks, you must store it in the `handles` structures.
8. Add this code to both *pushbutton1\_Callback* and *radiobutton1\_Callback*

```
handles.buttonCounter = handles.buttonCounter + 1;
set(handles.text1,'String',num2str(handles.buttonCounter));
guidata(hObject, handles); %without this line, the handles structure would not update,
```

%and the counter would not increment.

9. Now, save the .m file and run the GUI. Try clicking on the buttons to increment the counter. Notice that the counter increments when you activate AND deactivate the radio button.



## Sharing Data Between Callbacks and Sub Functions

It is good practice to make your functions modular so that your code is easily adaptable, robust, and flexible. Having said that, passing the entire handles structures to sub functions is something that I **don't** recommend. But there might be a time when you need to access the entire handles within a sub function. So here goes.

```
function [handles] = mySubFunction(handles)
%insert code here
%
%
```

This function will allow you to use the handles data within the sub function, and will also update any changes made within the sub function. You can add as many inputs and outputs as desired.

When you call this function, make sure to call it in the following manner:

```
[handles] = mySubFunction(handles);
```

This is the end of the tutorial.

# blinkdagger

an Engineering and MATLAB blog

- [Home](#)
- [Listchecker](#)
- [MATLAB](#)
- [Contact](#)
- [About](#)

## [MATLAB GUI Tutorial - A Brief Introduction to handles](#)

13 Nov 2007 [Quan Quach](#) [58 comments](#) 12,689 views

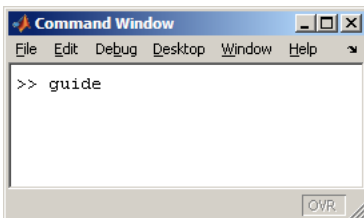
### Introduction

In this tutorial, we will discuss what *handles* are and how to use the get and set commands. When dealing with a Matlab GUI, you have probably noticed the variable *handles* being used in the accompanying .m file. The *handles* structure contains all the information for each object in the .fig file of the GUI. But it can also store other information. Let's explore a little bit more about *handles*.

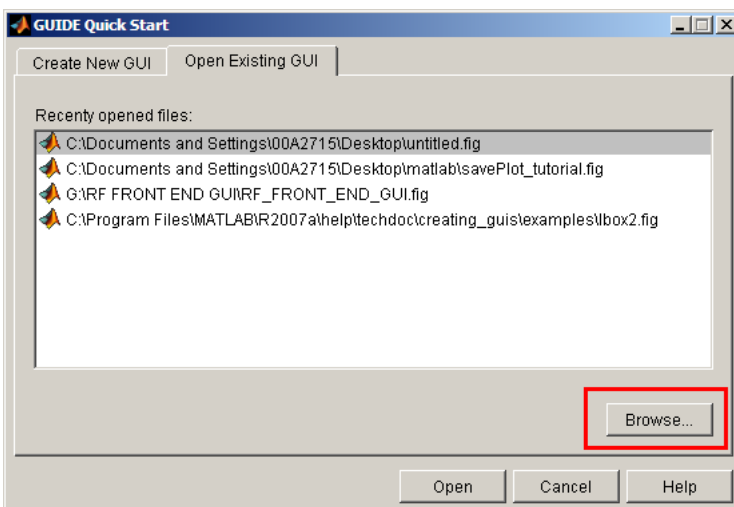
This tutorial is written for those with some experience creating a Matlab GUI. If you're new to creating GUIs in Matlab, you should [visit this tutorial first](#). Basic knowledge of Matlab is recommended. Matlab version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated).

### Handles, get, and set

1. First, download the sample GUI [here](#). Unzip the files and place them wherever you please.
2. Now, type guide at the command prompt.

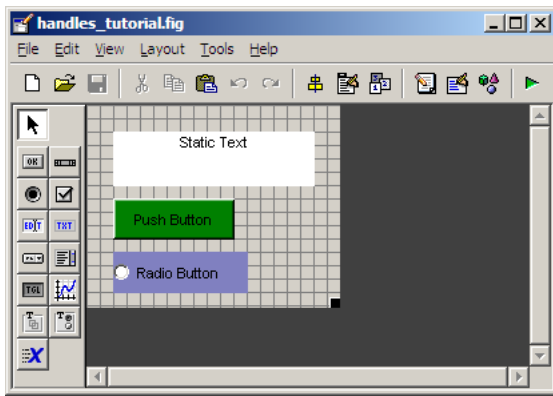



3. Choose to open the sample GUI by clicking on "Open Existing GUI". Click on "Browse" to locate where you saved the GUI files.

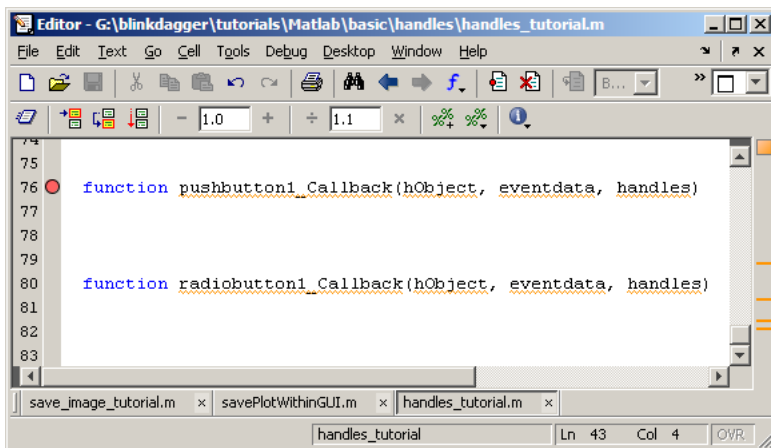


4. Here is what the GUI should look like when you open it:

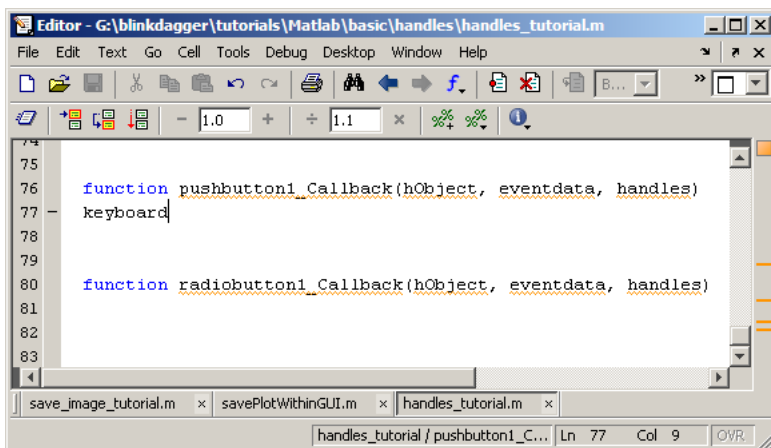




5. The handles structure contains all the information for the push button, radio button, static text, the figure itself, as well as output. So how exactly do we view and access this information? There are two ways to do this. Click on the  icon on the GUI figure to bring up the accompanying .m file. You can insert a breakpoint by left clicking on the left side of the .m file as shown.



Or you can type in keyboard into the .m file, as shown below.

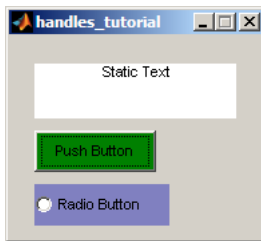


Both methods result in the same thing. This causes the GUI to go into command line mode, allowing you to examine and change the function's local workspace. (Incidentally, this is a great way to debug your code!)

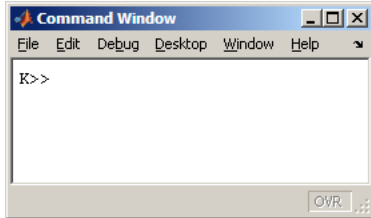
6. Let's use the *keyboard* method for this tutorial.

Type *keyboard* right below pushbutton1\_Callback, as shown in the figure above.

7. Now, save and run the GUI. Press the pushbutton.

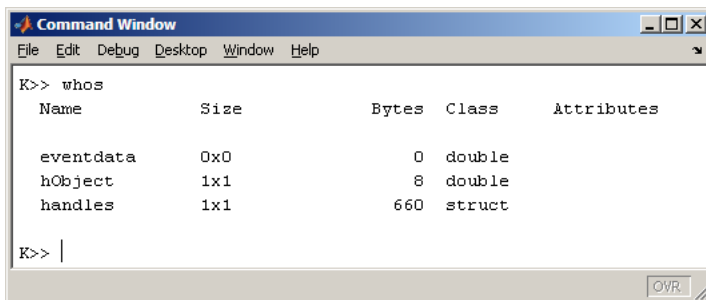


8. The command window should pop up and you should see the following at the command line. Notice the normal command line has been replaced with "K>>". This just means that you're in *keyboard* mode.

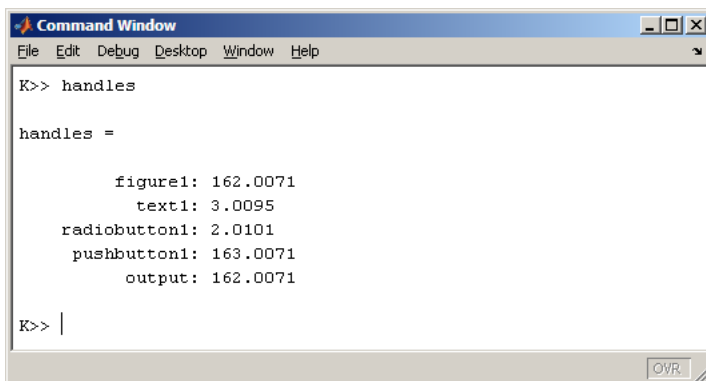


## Handles, get, and set (cont)

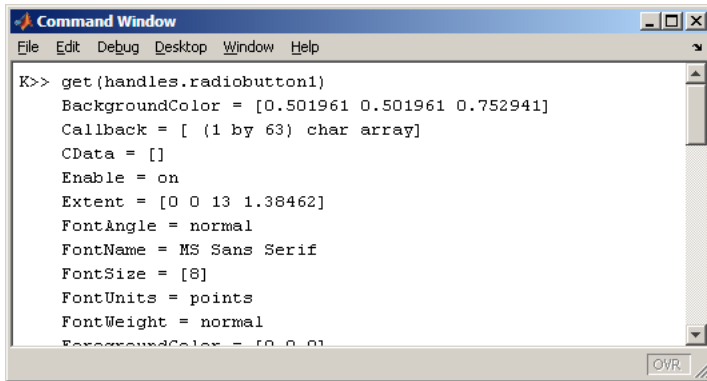
1. Now, let's take a look at the variables within the function's workspace. Type `whos` at the command prompt. This command tells you what variables are in the local workspace. Nothing too interesting.



2. Now, let's take a look at the handles. Type `handles` at the command prompt. This command gives you more details on the *handles* structures. You'll notice that each object on the GUI figure is accounted for. (figure1 is the background image that your components are placed on)

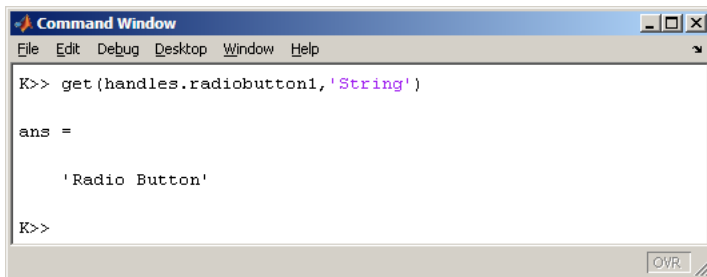


3. Lets say you wanted more details on the properties of *radiobutton1*. You can type `get(handles.radiobutton1)` at the command prompt to get a list of all the properties of this object. This command will display all the properties of that component, similar to what you would see in the Property Inspector when you double click on this component in the GUIDE figure.



```
K>> get(handles.radiobutton1)
BackgroundColor = [0.501961 0.501961 0.752941]
Callback = [ (1 by 63) char array]
CData = []
Enable = on
Extent = [0 0 13 1.38462]
FontAngle = normal
FontName = MS Sans Serif
FontSize = [8]
FontUnits = points
FontWeight = normal
ForegroundColor = [0 0 0]
```

4. Let's say you only wanted details on the *String* property for *radiobutton1*; you can type `get(handles.radiobutton1,'String')` at the command prompt. Additionally, you can store this value into a variable for later use. The `get` command is probably used most often with *Edit Text* components to extract user inputs.



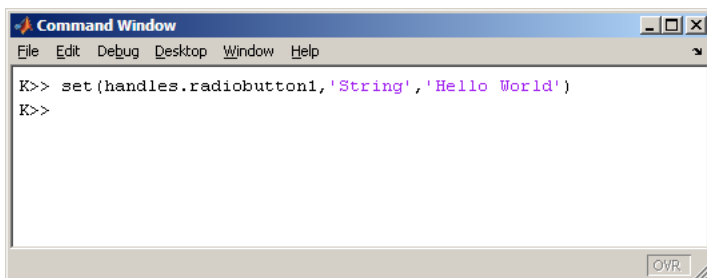
```
K>> get(handles.radiobutton1,'String')

ans =

    'Radio Button'

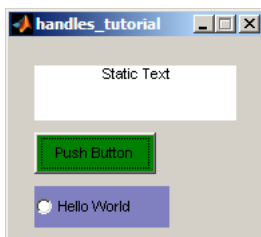
K>>
```

5. Lets say you wanted to change the *String* property on *radiobutton1*. You can do this by using `set(handles.radiobutton1,'String', 'hello world')` at the command prompt.



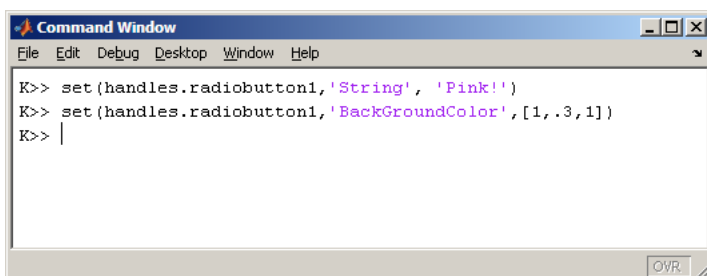
```
K>> set(handles.radiobutton1,'String','Hello World')
K>>
```

Notice that any changes you make using the `set` command are instantly reflected on the GUI program (not the GUIDE figure, but the actual GUI that is running).



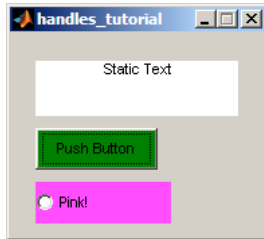
6. Try inserting these commands into the command line:

```
set(handles.radiobutton1,'String', 'The button is changed!')
set(handles.radiobutton1,'BackGroundColor',[1,.3,1])
```



```
K>> set(handles.radiobutton1,'String', 'Pink!')
K>> set(handles.radiobutton1,'BackGroundColor',[1,.3,1])
K>> |
```

The GUI that is running should now look like this:



7. After you're done playing around, type return at the command prompt to exit keyboard mode. Next, you should erase the keyboard command that you placed in the .m file and save it. Otherwise, the GUI will keep going into keyboard mode when you push that button.

