7.1 *MATLAB GUI basics*

MATLAB kindly[1] provides a tool (itself a GUI) for creating GUIs – the 'Graphical User Interface Development Environment' (**GUIDE**). **GUIDE** does 2 main things for you:

1. Firstly, it facilitates the design of the GUI window(s).
2. Secondly, it creates a code framework for the associated program.

You run **GUIDE** at the command line by typing its name:

```
» guide
```

and a window as shown in Figure 7.1 should appear. We'll only concern ourselves with the default option amongst the (4) 'GUIDE templates' – 'Blank GUI (default)'[2]. As for the tick-box 'Save new figure as:' – we'll leave this alone[3]. The 'Preview' window is blank at this point because you have selected a blank template (d'uh!) (and are not loading in a previously created GUI). So, all that remains, is to go ahead and click on 'OK'.

Actually ... before you move on, it is worth pausing at this point and reflecting on what happened and what the implications are for what you might like to do (GUI-wise). At the command line, you entered the command `guide`, which presumably ran a script or function (a piece of code in any case). A windows (the 'GUIDE Quick Start' window) was summoned (actually a figure window was created). The (figure) window did not open completely blank, but instead you might not:

- It has a close/minimize/maximize buttons at the top right (and the window can be re-sized).
- It has a title at the top (in the title bar) with a cute (barf) **MATLAB** icon.
- There are 3 buttons at the bottom right – 'OK', 'Cancel', and 'Help'. Presumably they'll all do something (different) when clicked.
- Everything else is neatly enclosed in a pair of *tabs* (one labelled 'Create New GUI' and one 'Open Existing GUI' and you can switch between tabs by clicking on the required tab.
- In the 'Open Existing GUI' tab, there is:
  - A list (of template names plus that annoying cute little icon again).
  - An area with a border labelled 'Preview' with a grey box labelled 'Blank' in the middle.
  - There is a *tick box* and next to it (grey-ed out by default), a box with a file path and name in and to the right of that, a

[1] For once, it is not a sperate, zillion-dollar license ...

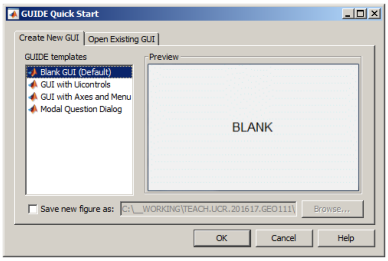

Figure 7.1: Starting GUI window of the **MATLAB GUIDE**, GUI design tool.

[2] So don't go randomly clicking on anything just yet!

[3] You can save the resulting figure (and code) under whatever filename you wish, later anyway. (If you really want, you can enter it in now here – it makes little difference.)

button labelled 'Browse'.

In essence, most of the primary (or at least, basic) features of a GUI are here to see. Funnily enough, nothing much had changed, at least in Windows, since ... the 80s[4]. Maybe that is a good thing as despite the **MATLAB GUIDE** tool being completely new to you, you hopefully can guess what would generally likely happen if you clicked on random bits of the 'GUIDE Quick Start' window.

(If you have not already clicked OK – do it now.)

Rather than creating some basic code first[5], MATLAB now throws you straight into a window design tool as per Figure 7.2. There is a lot going on here, but start by noting there is the usual drop-down menu bar at the very top (under the title bar ('untitled.fig') of the window) and a row of icon underneath that (no re-appearance of the **MATLAB** icon thankfully). At the bottom of the window there is some information, mostly about location (of what?). To the left of the window is a group of icons[6] plus a (depressed, by default) mouse pointer icon. Most of the window is made up of a pane (whose contents apparently is, or might be, larger than the area shown as indicated by the presence of scroll bars along the right and bottom edges). The pane itself is ruled with a grid pattern.

Again – the great advantage of familiarity (of program GUI design) – you might guess (you'd be correct if you did) that the icons to the left allow you to select an object and place it in the pane, the grid serving to help you position the object. And this leads us to an important point – creating GUI-based programs is as much (or more) about design as it is about programming. The cleverest program (and most complex calculations) might simply be a total fail if the GUI is wholly unappealing or complete un-intuitive (or lacks a GUI entirely). The grid is hence there for a reason and that is to guide you towards creating an ordered (and aligned), logical, and uncluttered arrangement of things (we'll come to what the 'things' are shortly) within the GUI window.

You might be tempted ... to click on everything and throw all sort of objects (what things?) into the pane of your embryonic GUI window. But the more GUI objects you have ... ultimately, the more code and the more debugging[7] you'll have to do. So we'll start as simply as possible and build up.

[4] That is: 1980s, as much as some might believe **Microsoft** has made little progress since the 1880s ...

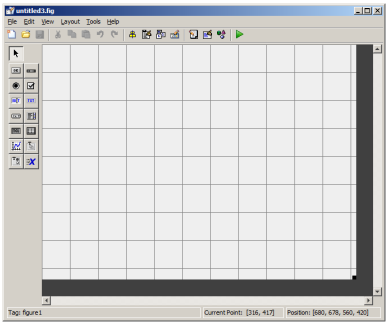

Figure 7.2: (Blank) GUI window editor GUI window.

[5] Actually, **MATLAB** has done this too and you would have seen it open up in the **Code Editor** window if you have provided a filename in the 'GUIDE Quick Start' window.

[6] Still no re-appearance of the MATLAB icon!

[7] Which has a steep power relationship with the amount of code.

7.1.1 Hello, World [Static Text (box)]

This is as simple as it is going to get for a 'program' with a GUI. In the GUIDE window editor already open, if you haven't fatally mucked about with it, or open up a new GUI by typing `guide` (or `GUIDE`) at the command line again – identify the Static Text icon (by

hovering the mouse pointed over an icon, its function is revealed). Click (L mouse button) on it. The mouse pointer, when over the gridded design pane, should change to a cross-hairs.[8] Find a convenient place .... perhaps at the intersection of two grid lines, click the mouse down and drag out a box – this will be the size (and location) of the Static Text object. Release the mouse button to finish. If you don't like the size or location, you can move/re-size just like you would to a **Windows** (or **MacOS** etc.) window.

So far, the (static) text object as a rather unappealing content of 'Static Text' in a pretty small font. You can edit the properties of this object by double-clicking on it[9]. Whoa! That's a long list of ... actually, *properties* of the object. Each property (the column on the left) has a default value (the column on the right) assigned to it. Evidently, you can edit the properties using the design tool rather than in the code code, setting a parameter value.[10] For now, we'll just make two changes:

1. For the String property – click in the box to the right, delete 'Static Text' and write 'Hello, World'.
2. The text is pretty small ... so for the FontSize property, click in the box to the right, delete 8.0 and write ... well, try something larger.

Within reason, play with some of the other properties if you like (at least, the ones that you can make a reasonably informed guess as ot what they do). Maybe you end up with a design window looking like Figure 7.3. Note that the effect of your changes is only shown if you e.g. hit Enter or click on a different property. If you accidently click outside of the text object an in the design pane, you'll end up switching the property editor to the window itself, which you don't want. (You can simply click back inside the text object to return the property editor to the text object's settings.)[11]

When you are done (editing properties) – click the Save icon. If this is a GUI that you have not previously created or previously assigned a filename to, you'll get a Save As dialogue box. At this point, **MATLAB** is going to save the window design with a .fig extension.

Something a little scary now happens – **MATLAB** opens up the code editor and some code, with a filename the same as you entered in but now with a .m extension. There is nothing we need worry about ... yet. And in fact, half the file is taken up with a main function that has the comment: `DO NOT EDIT`. Please take this advice ... :o)

Close the design window (and the code editor if it distracts you). At the command line, type the filename (no extension) to run the automatically generated code **m-file**. A window opens up ... the

[8] Note that this is to facilitate the positioning of the icon rather than being anything about guns and shooting at the coders behind **Windows**.

[9] I didn't actually read this anywhere – the operation of the editor or Windows has the same feel and intuitive usage as the sort (hopefully) of Windows you you are going to create in your GUI(s)).

[10] In reality: **MATLAB** is secretely writing the relevant code and setting the parameter value ...

[11] Unfortunately, the title of the property editor window is completely unhelpful – matlab.ui.control.UIControl when the text object properties are being edited, and matlab.ui.Figure when the (figure) window properties are being edited. So maybe watch out for Figure appearing in the title bar as an indicator or quite what is being edited.

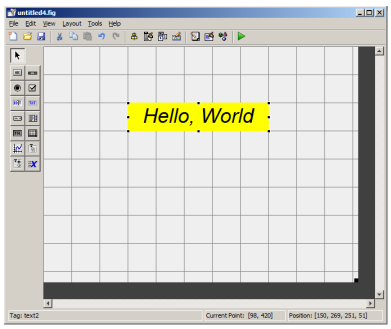

Figure 7.3: Design of the Hello, World window!

contents should come as no surprise, because you have just specified them (via the GUIDE GUI design tool). Your first GUI! But one you might notice does not actually 'do' anything – it just sits there unresponsive. Although you can at least close it (because it is automatically generated with the usual basic close/minimize icons plus the name of the m-file in the *titlebar*.

7.1.2 *Simple GUI responses [Push Button]*

A GUI is only of any particular use if it allows some response to input. This is going to involve a little code ... so we'll start with the simplest possible action – a *button* that performs a simple action (closes the window).

Re-run `guide` and open up a new window editor (by clicking OK in the GUIDE Quick Start window). Now find the Push Button icon, click it, and drag out a push button object in the design pane. You should see a box (with a pseudo 3D shading at the edges) with the text Push Button in the centre as per Figure 7.4. As before, you can edit the properties of the push button object (because the default properties are totally boring) by double-clicking it. Start by editing the font (size) and message. Perhaps 'Go away!'. And save ...

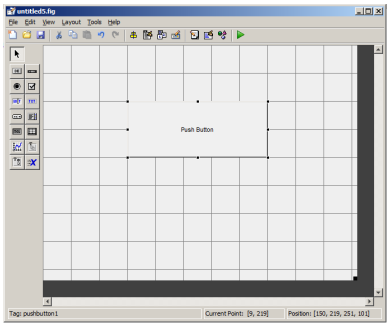

Figure 7.4: Design window with a default push button object.

When it saves, MATLAB again opens up the code it generated. There is slightly more code in the file this time and shortly, we'll need to look at it. But for now: type the name of the file at the command line. You'll get a window opening with the push button you created in it. Click on it. It does seems to 'respond' (pretends to depress by means of changing the edges with the pseudo 3-D shading) to the mouse click, but ... nothing else happens. This is where YOU (and your amazing coding skills) now come in.

If you have closed the design window, re-run `guide` and rather than creating a new GUI – switch to the Open Existing GUI tab and double-click your filename (of the push button GUI) or select and OK. Double-click on the push button object to open up the property editor. We'll make only one (more) change here – down the list of properties your fine 'Tag'. This is the name (ID or *handle*) of the push button object.[12] By default, the name is pushbutton1. Edit this to ... goawayButton (or pick an alternative name) and re-save the GUI.

[12] In essence, no different from a filename – a unique identifier for an object (/file).

Go to the code editor for the associated **m-file** (which will have the same name remember). In the file we have:

- The main function which we can ignore (and indeed apparently should not be edited!).
- `function goaway_OpeningFcn` which is executed when the GUI is started up. This is the place to put code for initializing models or whatever.

- I have no idea what `function varargout = goaway_OutputFcn`. Textbooks helpfully say to ignore this. Great idea.
- Finally, `function goawayButton_Callback`. This function is executed when your 'Go Away!' push button is pressed.

In this simple GUI, we have only one figure and it is active (has the mouses' attention), so we could simply use the `close` command ('*deletes the current figure*'). Insert this simple command in the `function goawayButton_Callback` function, after the last comment line.[13] Save the **m-file** and re-run. Now if you click on the 'Go Away!' push button, the window does indeed go away (aka, closes).

[13] Note that automatically generated MATLAB code does not seem to ever formally `end` a function as one really should do ...

7.1.3 *Updating object properties (do you like bananas?)*

Bananas. Do you like them? Perhaps the GUI can provide an answer (rather than just text statements written to the command line via `disp` as before).

Now you are going to want to think about the design of the GUI a little. What we want is for the the GUI to display a question ('Do you like bananas?'). There will be two options, 'Yes' and 'No' that can be clicked. Depending on which one is clicked, some appropriately supportive, or otherwise, message will appear in response. We need:

1. A plain (static) *text box* as before to display the question.
2. A pair of *push buttons* (again as before).
3. Another plain (static) *text box* to display the answer/response.

And ... we are going to need some code that, depending on which button is pushed, displays a different message.

The latter part is not as bad as it sounds. We could have no test initially in the 2nd (static) *text box*. We just need to change its text property (i.e. change the no text to our message). This is mostly a case of working out and using the unique identifier of this text box object AND the identifier of the text property (of the text box object).

Firstly – re-run GUIDE. Create a new GUI window with the 4 elements (2 static text boxes and 2 push buttons). It is up to you how you arrange these 4 objects in the design pane. You might be guided how windows in programs you have used are designed. AT the minimum, it is standard practice to place a 'No' push button next to and aligned horizontally, with the 'Yes' (and often 'Yes' to the right of 'No').

No idiot would design anything like Figure 7.5 and certainly not with those color choices ... but you get the idea.

For each of the objects (2 text boxes and 2 push buttons), I have renamed them (the Tag property) to something more memorable than e.g. button or box, #1, #2, #3, etc etc..

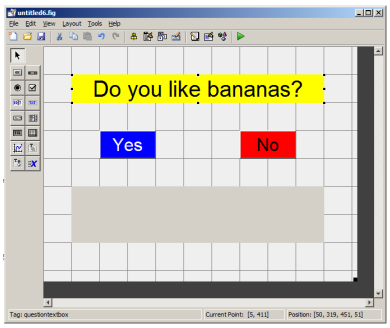

Figure 7.5: (completely) Bananas design window.

The code that MATLAB generates for `bananas.m` (my name) is not a lot more involved than before. Primarily, there is just a second function associated with a mouse click on the 2nd push button.

The logic is going to be very simple. In fact, we don't need any, because if the Yes button is clicked, **MATLAB** will call one function (my name: `function yesbutton_Callback`), and if the No button is clicked, the other function (`function nobutton_Callback`) is called. As alluded to above, how do we get the text to change in the 2nd text box (from the default of no text)?

Unfortunately, **MATLAB** get all weird here.[14] If you had a friend called Luna, you might reasonably communicate with them via the name 'Luna'. MATLAB doesn't do it this way and instead assigns a numeric ID. Luna might have an ID of 8.206034. So you are going to have to get this ID, which in this case is the ID of the 2nd text box, if you want to change a property (here: the displayed text).

[14] Actually, no wierder than **netCDF**. Or arguably **Python** ...

First off, you can get the ID of the object property using `findobj` and assign the result to some memorably variable, e.g.

```
h_answertext = findobj('Tag','answertextbox');
```

This is as simple(!) as asking to find the ID of the object which has a `Tag` with value 'answertextbox' (which was the value I set in the design editor).[15]

[15] What we might refer to as an ID, **MATLAB** calls a *handle*. Hence commonly an 'h' might appear at the start of a variable name to indicate it contains a *handle*.

Where would we put this line of code? Why not in the initialization function, `function bananas_OpeningFcn` and I am guessing, at the end of that function[16].

[16] Be careful as MATLAB is not automatically adding an `end` to the end of functions ...

Now – we have the ID of the 2nd text box and we can now set its property (from no text t a suitable message). Lets first implement an answer if the Yes push button is clicked. The command to set a property is ... `set`. In our example, the handle we have already obtained and assigned to the variable `h_answertext`. The name of the property we want to change (refer to the column list in the property editor if you like as a reminder) is `'String'`. And the text ... well, you can have whatever you want. The complete line is then:

```
set(h_answertext,'String','Yes, it is an excellent fruit.');
```

`set`
Sets ... the property value of an object. The syntax is:

```
set(h,name,value)
```

where `h` is the handle (the ID obtained via `findobj`), `name`, is the name of a property, and `value`, the value of a property.

Well, it turns out that this does not quite work? Why? My guess that we could add the line:

```
h_answertext = findobj('Tag','answertextbox');
```

to the initialization function was incorrect. One possibility is that the function states:

```
% -- Executes just before bananas is made visible.
```

meaning that this function is called before the window is opened. If the objects have not yet been created at this point, they will obviously

have no ID and the variable `h_answertext` would be empty. Alternatively, the variable `h_answertext` created in the initialixation function is simply not accessible (visible) to `function yesbutton_Callback`. Sod it, the line has been added to `functionyesbutton_Callback` instead, giving:

```
h_answertext = findobj('Tag','answertextbox');
set(h_answertext,'String','Yes, it is an excellent fruit.');
```

Now it works and creates the result shown in Figure 7.6.

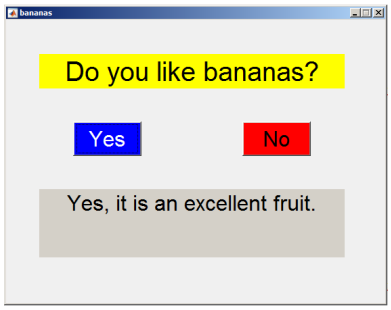

Figure 7.6: (completely) Bananas GUI in action.

Now extend this so that an alternative answer is provided if the 'No' button is instead clicked. Other embellishments you could make might be to make the color of the button you clicked change. This is simply a matter of finding its object ID, and setting the property BackgroundColor.

Finally, and to put a little of your coding skills to the test, how about displaying a 3rd message ('Make up your mind!'?) if someone changes their mind – i.e. if a second button is pressed (after the first). You'll need a variable to store whether any button has been pressed and assign this an initial value of false, e.g.

```
var_pressed = false;
```

Whenever a button is pressed, `var_pressed` will become (will be set to) true. So before displaying the message in both of the button press *callback* functions, the value of `var_pressed` needs to be tested – a false means this is the first time any button has been pressed. Once that initial message is displayed, the `var_pressed` becomes true, and when the next time a button is pressed and the value of `var_pressed` tested, a *true* leads to a different message. All that is needed is an `if ...` in each callback function, and a line initializing `var_pressed` to false (in `function bananas_OpeningFcn`). There is just one problem ...

Variables in functions are 'sectret' (*private*) and limited (in *scope*) to just that function. So the variable `var_pressed` which you initialized at the end of `function bananas_OpeningFcn` cannot be seen by the callback function.

We can enforce that the same variable is seen by multiple functions by stating that it is *global* (in *scope*):

```
global var_pressed;
```

This line needs to appear at the start of each function in which you need to read or write the value of `var_pressed`, i.e. in both callback functions as well as the initialization function. The complete code for the Yes button call box function would then look like:

```
% -- Executes on button press in yesbutton.
function yesbutton_Callback(hObject, eventdata, handles)
% hObject handle to yesbutton (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see GUIDATA)
%h_answertext = findobj('Tag','answertextbox');
global var_pressed;
h_answertext = findobj('Tag','answertextbox');
if ~var_pressed
    set(h_answertext,'String','Yes, it is an excellent fruit.');
else
    set(h_answertext,'String','Make up your mind!');
end
var_pressed = true;
```