APPENDIX B

GUI DEVELOPMENT

B.1 INTRODUCTION

In this appendix, we provide a practical guide to develop graphical user interfaces, or GUIs, for MATLAB applications. This walk-through will give you the general idea of how a user interface works in MATLAB, from laying out the window in code to passing data around your application. We will refer to an example GUI throughout the appendix so that you can see the concepts in action.

First, we will look at the basic structure of the code that makes up a GUI. This is important because the structure of the code is related to its function, as we will see later. We will also take a look at how the control of the system is passed between MATLAB and the functions that make up a GUI. This will allow you to implement a fancy dynamic application, even though MATLAB executes code sequentially. Next, we will inspect the method by which data are saved in the GUI and how those data are passed around so that all necessary functions have access to them. Finally, we will dissect a working GUI demo that will wrap up all the concepts covered as well as solidify them.

B.2 GUI FILE STRUCTURE

The file that makes up a GUI interface is nothing more than an M-file. The typical uses of an M-file you have most likely seen so far are to save code in a script or create

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

a stand-alone file, or a function, that performs a particular task. A GUI M-file is a lot like a function in that it performs one task: create and execute your GUI. It differs, however, from a function in the structure of the file. When writing a function, you have the option of writing subfunctions to take care of operations accessible only to your function. In a GUI M-file, there will always be subfunctions present, sometimes many. The basic structure of a GUI M-file is shown in the pseudocode that follows:

```
function mygui(parameter1, parameter2, ...)

if no parameters were passed, then
    initialize();
else
    switch over the parameters
        case 1:
            subfunction1();
        case 2:
            subfunction2();
        ...
    end
end

function initialize()
(code to initialize goes here)

function subfunction1()
(code for function1 goes here)

function subfunction2()
(code for function2 goes here)

...
```

The first line contains the function definition as usual. We will take a look at the uses of the parameters later.

Let us analyze the subfunctions. Three subfunctions are shown in the model, but more can exist (and usually will). The first subfunction is uniquely called `initialize()`. This should always be the first subfunction. It will initialize the GUI, as the name suggests. More specifically, this subfunction will create the GUI figure and generate all the buttons, axes, sliders, and any other graphical items necessary. It will also load any default parameters your application may contain. For example, if your application allows the user to blur an image dynamically by dragging a slider bar, you may want to start out with an initial amount of blur. Even if you wanted no blur initially, this must be defined, and you would do it in the `initialize()` subfunction. The initialization subfunction is executed only once—when the GUI runs for the first time. After that, it is up to all the other subfunctions to make sure all goes well.

The name of the initialization subfunction is not required to be `initialize()`, but this name is easy to understand and is self-explanatory. Ultimately, it is up to you how you wish to name it.

The other subfunctions, depicted as `subfunction1()` and `subfunction2()` in the model, represent all the other functionality that your GUI application will contain. Any action that could take place within the GUI will be coded in its own separate subfunction. So, when a button is pressed, there is a subfunction that handles that action. Same goes for sliders, pull-down menus, or any other interactive elements. Subfunctions can also be used for repetitive tasks. For instance, if there are many interactive elements in your GUI having a common component in their code, it is wise to factor that code out and place it in its own subfunction, thereby reducing the complexity of the code and making your code easier to maintain.

So far we have looked at the subfunctions of a GUI M-file, which have three basic uses: initialization of the GUI, encapsulating the functionality of interactive components in the user interface, and factoring out reusable code. It seems as if we have put all of the code into subfunctions, so you may ask if there is a use for the main function itself (in the case of the model, `mygui()`). The main function has a small but very crucial role in the execution of a user interface. Let us first look at what happens when this code is executed and later we will see why it is all necessary.

The only aspect to the main function is an `if-else` statement. Basically, the functionality is as follows: if no parameters are passed when the function is called, then its only job is to initialize the GUI. If parameters are passed, then it performs a switch in such a way that different combinations of parameters will give access to the subfunctions. That is it! That is the concept of the main function. You can think of it as a hub—from the main function, we can access all other subfunctions. As you will see later, this is what makes dynamic execution possible in MATLAB.

We will come back to our model later to see exactly how its structure meets its function. To continue, we must look at how control is passed in MATLAB.

B.3 PASSING SYSTEM CONTROL

One of the main challenges in developing a GUI in MATLAB is to give the user the feeling of complete control over the application, just as in any other program. By this we mean that the user can interact with the interface in a non-predetermined way, and the application will still execute as expected. This is a small problem in MATLAB because code is executed in a top-down fashion, meaning when a script or function is called, execution begins at the top of the file and works its way down and finally exits. If we look at the system control diagram depicted in Figure B.1, we can see this is consistent. When users run a script, they do so by calling the file name or a function name. Control is given to that function and when the function is complete, control is given back to MATLAB. You may have noticed that when running functions or scripts that take some time to execute, MATLAB is not responsive until that function is

complete. This is because the function has control, not the main interface of MATLAB. Coming back to our problem, we can see that this type of execution is not consistent with a dynamic environment where the user may click on components in random order. By understanding how control is passed in MATLAB, we can understand the clever design of a GUI M-file that allows a dynamic environment.

When analyzing the passing of system control, keep in mind that the structure of the GUI M-file, as you will see, is consistent with the concepts depicted in Figure B.1. When MATLAB is first started, the user is presented with a command prompt. To start a GUI application, the user types the function call for that GUI, and it is loaded. From here, MATLAB passes control from the command prompt over to the main function (shown as step 1). Let us assume that the user did not pass any parameters to be consistent with our M-file model. Because there were no parameters, the main function sends control to the initialization subfunction (step 2). As stated previously, this subfunction creates the GUI window and all other components (step 3). Once the initialization subfunction is complete, control is returned back to the main function (4), but there is no more code to execute here (remember the main function was only a hub). So from here, the main function ends and control is returned to MATLAB (exit). Although control is returned to the command prompt, we now have another window open in MATLAB, our GUI. MATLAB is aware of this, and if we interact with any components, MATLAB will know.

It is probably a good time to introduce the concept of a callback. A callback is simply a line of code that executes when someone interacts with a component of your GUI. Every component has its own callback, and it is defined when the GUI is initialized. Remember that this code is small, so we cannot put the entire functionality of the component within the callback, but just one line of code that will call a function containing the real code that implements the component. So, to summarize the concept of a callback, the following events would take place: when the user interacts with a particular component (i.e., a button), MATLAB grabs the value of the callback for that component and then executes it as if it were typed into the command prompt. Now, because this code must be small, we want to design our callback in such a way that we will ultimately execute the code that gives our component its functionality.

If we observe Figure B.1, we see that when a user interacts with a component, MATLAB will execute the callback for that component (which is just one line of code). But, if you remember, the implementation for that component is one of the subfunctions of the M-file. This is a slight problem because subfunctions are only accessible from within the M-file. Remember, we are no longer executing code in the M-file, so currently we do not have access to that subfunction (i.e., MATLAB currently has control). To overcome this issue, we will define the callback such that it will again call the main function of our GUI M-file, but instead of calling with no parameters, we will specify the specific parameters that will gain us access to the subfunction we need. This is the heart of GUI functionality. Keep also in mind that each component has its own unique callback code, and therefore we can define unique ways of calling the M-file such that all the subfunctions are accessible. Once that callback code is executed (shown as step a), we are again inside the M-file. The

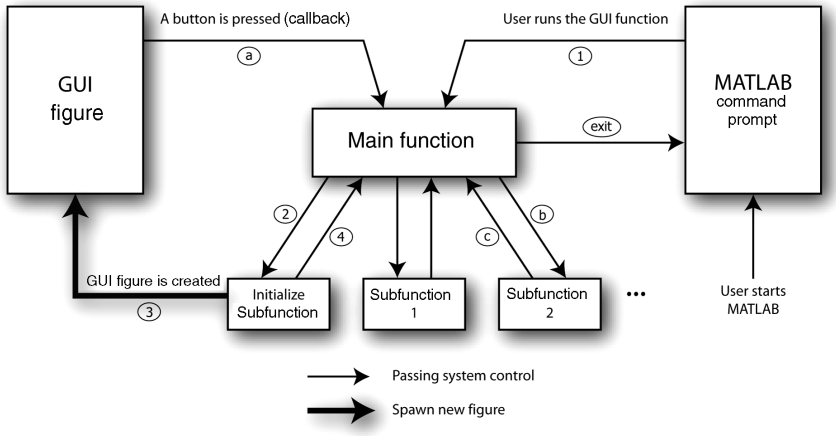

FIGURE B.1 System control diagram.

first chunk of code that is encountered is that hub-like `if-else` code. Here, we are forwarded to the appropriate subfunction (step b), where it does its business and then returns control to the main function (c). From here, we again see that there is nothing else to execute in the main function, so the function ends and control is returned to MATLAB (exit).

Once the GUI figure is initialized, the process of executing callbacks is repeated every time the user interacts with the GUI (following the sequence a–b–c–exit). The only time this stops is when the figure is closed. At that point, the only way to get back to the GUI is to start it up again by calling the main function.

So now we can see how the structure of the GUI M-file meets its function. The M-file must be structured so that the entire appropriate code is accessible by the GUI figure. Although it is nice to understand how this all works, you will not get far with a GUI unless you have a clear picture of how data can be saved, accessed later, and manipulated. Thus, we come to the `UserData` object.

B.4 THE USERDATA OBJECT

As you will see, the concept of the UserData object is actually quite simple. Before we have a look at it, let us first take note to a small feature of MATLAB. The variables available for use are organized in a stack, as illustrated in Figure B.2. This means that when MATLAB passes control from the command prompt over to the main function, any variables that were available are no longer accessible. They still exist, but they are accessible only from the command prompt. Similarly, when control is passed to a subfunction, any variables that might have been present in the main function are no longer available. Also, if variables are created within the subfunction and once we exit the subfunction and control is passed back to the main function, we would not

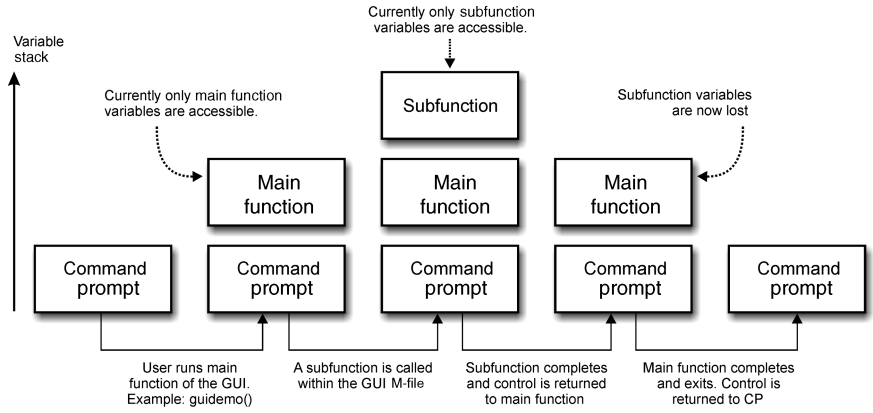

FIGURE B.2 Variable stack.

have access to those variables that were once accessible from the subfunction—those variables are in fact lost forever!

What this means for us in terms of a GUI is, at this point, we have no way of saving any data. If you notice, from the system control diagram, the main function is constantly called, but every time it exits, all variables are lost—the function is brought to life and killed constantly and right along with it follows its variables. In this entire GUI environment, there is in fact only one item that is always in existence other than MATLAB itself and that is the figure of the GUI. Because of this fact, we will store any persistent data within the GUI figure. The data structure that houses this information is traditionally known as the `UserData` object.

The `UserData` object is a child of the GUI figure and usually goes by the name `ud`, although technically the object could be named anything you want. The `UserData` object is actually just a variable, and as all other variables in MATLAB, it can take on any form that a variable may express. If the only persistent data in your application is an image, then `ud` can be equal to the image data. If your application holds different pieces of unrelated data, then `ud` can take on the form of a structure that is the most common. This is by far the most flexible because future changes may require an addition of information to the persistent data in your GUI and by initially taking on the form of a structure, this information can easily be added to the structure (remember a structure can contain elements of different types or classes).

With regard to the `UserData` object, the only thing left to look at is exactly how we initialize it, grab its contents when needed, and how to save back to it. This will be covered when we inspect the demo GUI in the next section.

B.5 A WORKING GUI DEMO

You should notice that the demo GUI M-file (available at book's web site) looks similar to our model pseudocode from earlier. There is one parameter that will be

used by the GUI during run-time, and based on the switch statement, it can take on a value of 'average' or 'complement'. The initialization subfunction accounts for the majority of the code for this demo, so let us take a look at its code first. Remember that this is a simplified demo, and there are many other creative things that can be done in a GUI file; these are just the basics.

If the user runs the GUI function (simply by typing `guidemo` at the MATLAB command prompt), the `if-else` statement will direct system control over to the initialization subfunction. The first block of code will determine whether the GUI is already running. This is achieved by first searching MATLAB's list of objects for the name of the GUI. If it is present, we will close it. Of course, in your own application you can do whatever you want. Your functionality may allow multiple instances of the same GUI application running simultaneously. It is up to you and the design of your application.

The next block of code creates the GUI figure and sets its properties. After this, the figure is resized and relocated to the center of the screen. Next, all the components on the figure are created. You will notice, for example, that when the 'average' button is created, we are also saving its handle in the `UserData` object. A handle is simply MATLAB's language for a pointer to that component. We do this so that we may have access to the button and its properties if needed later in the application. Also, notice that the callback property of the 'average' button is set to `guidemo(''average'')`. Here, we are telling MATLAB to execute the `guidemo()` function with the parameter 'average' every time this button is pressed. It is important to know that the quotes around 'average' in this statement are two single quotes, *not* double quotes!

The last two components added to the figure are axes. Remember that an image is rendered on an axes—this is what these two axes are for. In the final steps of the initialization subfunction, we load the default image and save it to the `UserData` object. We then display the original loaded image in the `org_axes` axes. We could just stop there and let the user decide which filter to use by pressing one of the buttons, but at this point there is nothing rendered on the second axes, and so it will display as an axes. This might not be the best way to present the application when first loaded, so we execute the code to filter with an averaging filter by calling the appropriate subfunction. Notice that this is indeed the same function that is executed when the user presses the 'average' button and this is just fine.

The last two subfunctions implement the two buttons. Both for the most part are identical, except for the filtering part, which is not of importance here, so we will look at only one of them. In the `filter_average` subfunction, the first thing we do is get the `UserData` object from the figure by using the function `get()` and save it in our own variable `ud`. Remember that this variable will be gone once we leave this subfunction, so we should remember to save any changes back to the `UserData` object in the figure before exiting. The next, block of code simply filters the original image and saves this new image in an appropriate variable inside our copy of the `UserData` object. Next, we display the filtered image. Notice that we must first select the appropriate axes by using the axes function. If this code is not placed before the `imshow` function call, MATLAB will display the image in the first axes it finds, which in this case would most likely be the original image axes. Finally, because we

have altered our version of the `UserData` object, we save it back to the figure by using the set function `set()`.

B.6 CONCLUDING REMARKS

Remember that this M-file was just for demonstration purposes and most likely your GUI M-files will be more complex to achieve your desired functionality. In this demo, we did not allow the user to pass any parameters to the function when calling it, but this could be done by modifying the `if-else` statement to allow it. Also, each component that can be used in a GUI has many features that allow you to customize the GUI, so it is a good idea to study the different components and their capabilities before designing your GUI. With your understanding of what goes on behind the scenes when a GUI is in execution, you should be able to design and implement a GUI that will fit the needs of your next MATLAB prototype.