MATLAB APP Components & Properties:

Typing the command appdesigner in the Command Window will bring up the App Designer. Fig. 1 shows the App Designer; in the middle is the blank layout under "Design View." The Component Library on the left shows the icons of the components that can be created. On the right are the Component Browser and Component Properties Windows. In the Component Browser, the default name for the blank figure, app. UIFigure, can be seen.

```
>> appdesigner
```

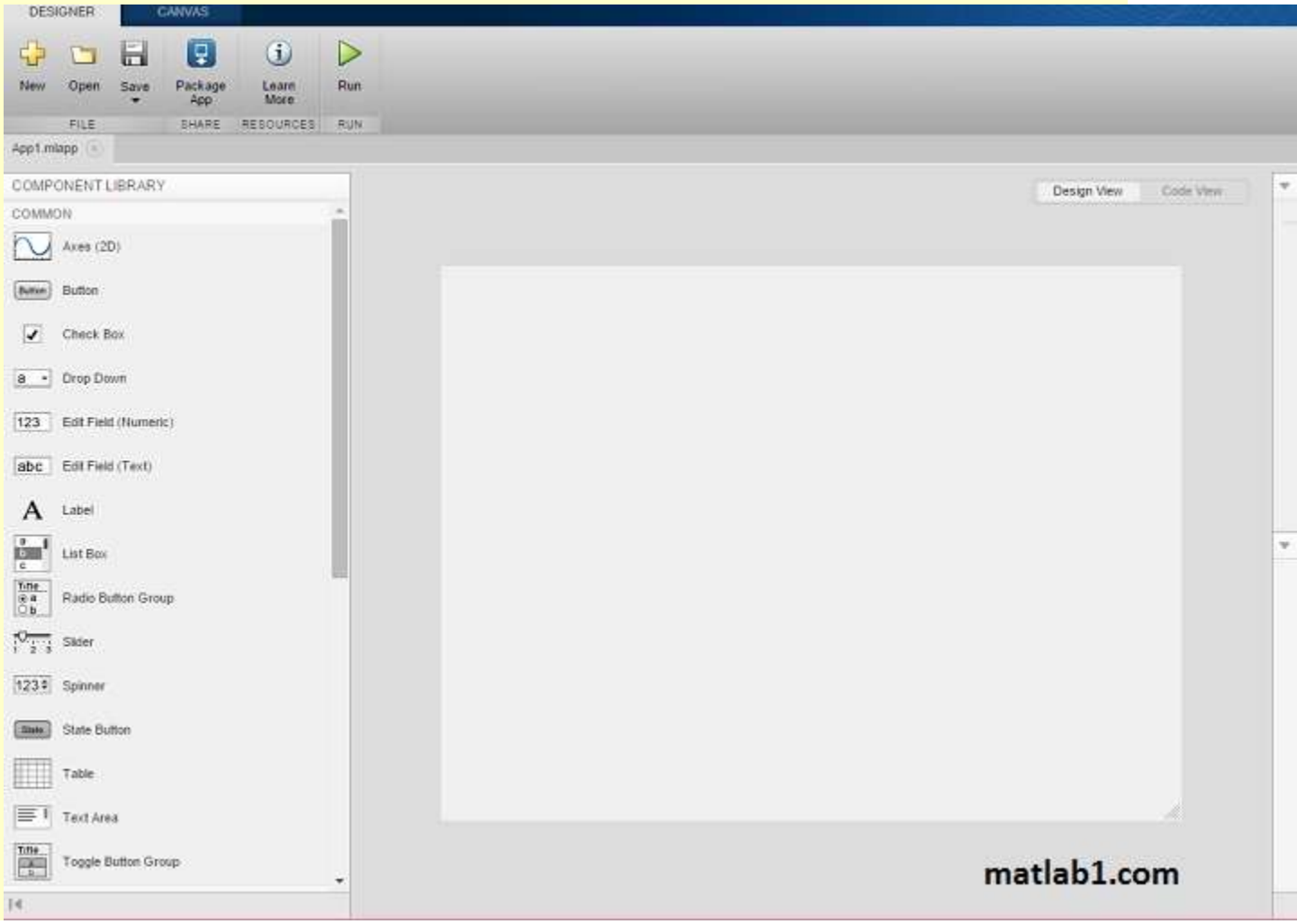
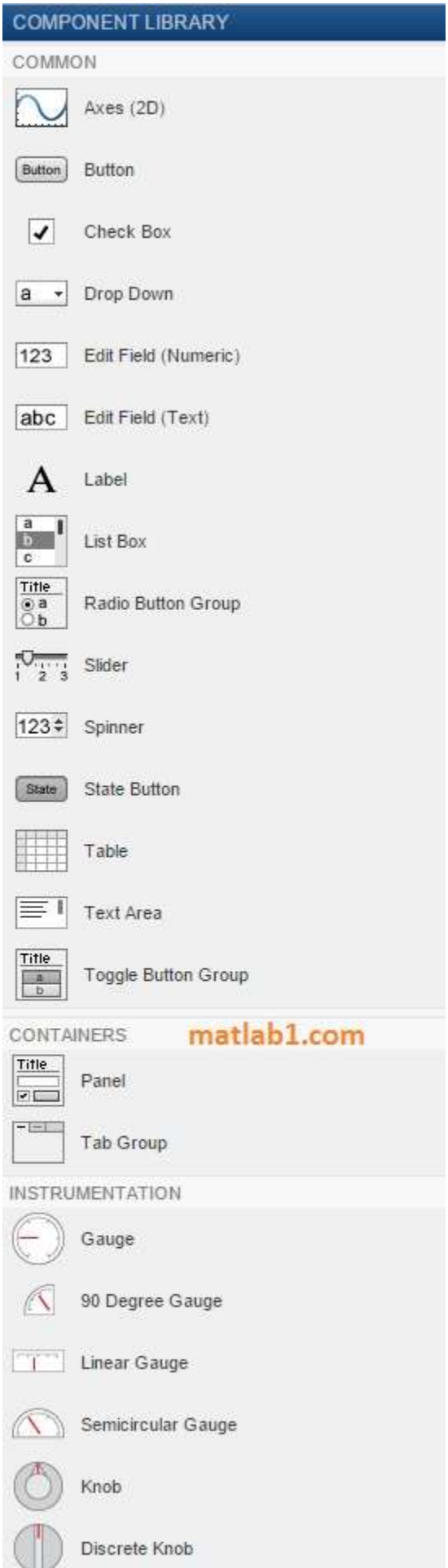
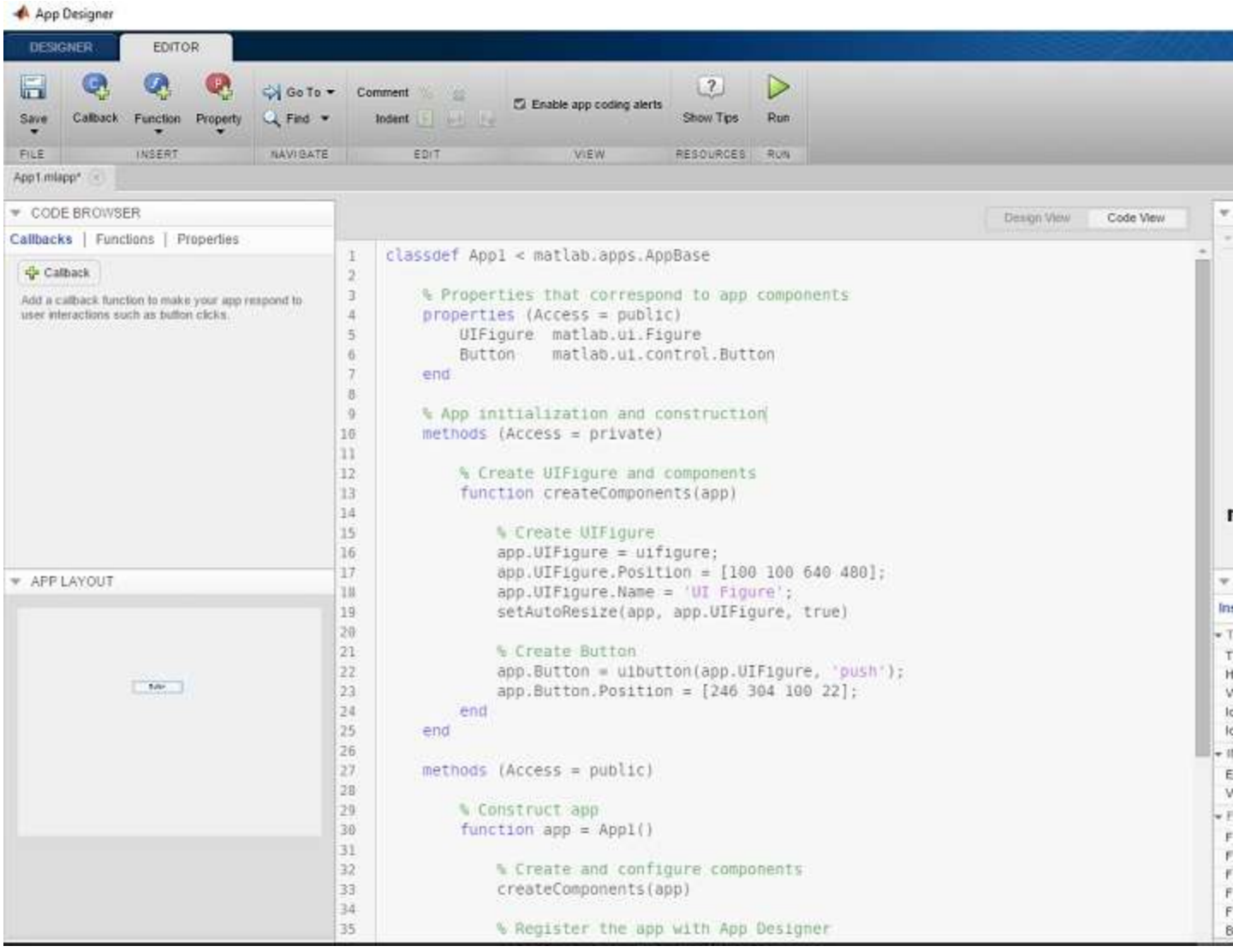


Fig. 2 shows the remainder of the Component Library, including the Container and Instrumentation components. Clicking on “Code View” instead of “Design View” shows the code that has been created for the blank app. App Designer creates a class named Appl that is derived from a MATLAB apps superclass called matlab.apps.AppBase. The properties will consist of all of the components; for now, there is just one property, which is the UI Figure Window. There are also two private methods blocks and one public methods block.


COMPONENT LIBRARY
COMMON
Axes (2D)
Button
Button
Check Box
Drop Down
a
Edit Field (Numeric)
123
Edit Field (Text)
abc
Label
A
List Box
a
b
c
Radio Button Group
Title
a
b
Slider
1 2 3
Spinner
123
State Button
State
Table
Text Area
Toggle Button Group
Title
a
b
CONTAINERS
matlab1.com
Panel
Title
Tab Group
INSTRUMENTATION
Gauge
90 Degree Gauge
Linear Gauge
Semicircular Gauge
Knob
Discrete Knob

The methods are used for all of the app functions, including eventually callback functions. For now, with just a blank UI Figure Window, the public methods include the constructor function Appl, and a function delete that deletes the figure when the app is deleted. The constructor calls a private function *createComponents*, which creates a UI Figure using the *uifigure* function. It then uses the dot notation to set properties for the app. *UIFigure*, including its Name and Position. Note that the dot notation is used twice here, as in *app.UIFigure.* Name.



The function *startupFcn* executes when components are created. It is called by a function in the constructor. Within the Code View, most of the window is grey, which means that the code cannot be modified. However, within the *startupFcn* function there is a white box, which means that code can be inserted in that function. Fig. 3 shows the *startupFcn* function in Code View.

```
methods (Access = private)

    % Code that executes after component creation
    function startupFcn(app)

    end
end
```


We have seen the code that is generated with just a blank canvas. In the Design View of the App Designer environment, components can be dragged from the Component Library onto the blank layout. When this is done, the code will be updated to create a new property for each component, and to initialize some of its properties (for example, the Position property will be based on the location to which the component was dragged). Properties can then be modified in the Component Properties Window to the right of the layout.

For example, dragging a Label (generically named ‘Label’!) into the design area causes the Design View to look like Fig. 4.

Clicking on the label itself in the Design View will change “Component Properties” to “Label Properties.” Properties such as the text on the label, its justification, font name, font size, and so forth can then be modified from the Label Properties window. Each of these will modify the code that is created.

Assuming that no properties have been modified, and there is just a generic label in the layout as shown in Fig. 4, most of the code remains the same as with just a blank UI Figure Window. Choosing Code View will show that the class App 1 now has a new property Label, and the *createComponents* method creates a Label using the *uilabel* function and gives it a Position. The added code is shown in bold here (not in App Designer itself).

```
classdef Appl < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access= public)
        UIFigure matlab.ui.Figure % UI Figure
        Label matlab.ui.control.Label % Label
    end
% Create UIFigure
  function createComponents (app)
```

```
      app.UIFigure = uifigure;
      app.UIFigure.Position = [100 100 640 480];
      app.UIFigure.Name = 'UI Figure' ;
      setAutoResize(app, app.UIFigure, true)
      % Create Label
      app.Label = uilabel (app. UIFigure);
      app.Label.Position = [421 431 30 151];
   end
end
```

Modifying a property, such as changing the text of the label in the Label Properties window to be

"Hello" instead of "Label" will modify the code in

createComponents to:

```
% Create Label
 app.Label = uilabel(app.UIFigure);
 app.Label.Position = [421 43130151;
 app.Label.Text = 'Hello';
```

It is also possible programmatically to modify properties such as app. Label. Text in

the *startupFcn* function, but it is preferable and easier to do this from the Design View.

To run the app, click on the green Run arrow. This brings up a dialog box that asks for the name of the file that will be created; the default name that shows is "App 1.mlapp." Changing the name to "HelloLabel.mlapp" will create a file with this name and bring up a UI Figure Window. The upper right portion of this is shown in Fig. 5.

Note that the App Designer creates one file with the extension '.mlapp,' unlike GUIDE, which creates two files: a .m file and a .fig file. While this example shows the basics of App Designer and the object-based code that it creates, it did not involve any callbacks.

UI Figure Functions

Traditional GUIs create different types of objects by specifying the Style property in the *uicontrol* function. For example, a slider is created with *uicontrol('Style', 'slider')*. By contrast, App Designer uses separate functions to create the different component types; for example, a slider is created with the *uislider* function. In the previous section, we saw two of these functions: *uifigure*, which creates the UI Figure Window, and *uilabel*, which creates a static text box.

All of these functions can be called directly from the Command Window or from any script or function. For example, the following code will bring up the UI Figure Window shown in Fig. 6.


```
>> uif = uifigure;

>> uif .Name= 'Simple UI Fig Slider';

>> uislid = uislider(uif);
```

By creating a UI Figure Window first, and then passing its handle to the *uislider* function, the slider is put inside that UI Figure Window (otherwise, a new UI Figure Window would be created for the slider). The Value property of the slider can be inspected, e.g.,

```
>> uislid.Value
   ans=
       0
```

Of course, to do anything with the value of the slider would require more complicated code with callback functions. For sliders, there are two:

- *ValueChangedFcn*‘: executes when the slider value has been changed
- *ValueChangingFcn*‘: executes as the slider is being moved

For example, the function *uislideruilabel* creates a UI Figure Window with a slider and a label. When the slider has been moved, the *ValueChangedFcn*‘ callback is called, which puts the value of the slider in the label as seen in Fig. 13.35. Many of the concepts here are similar to those used in GUI callbacks.

The function *uislideruilabel* is a function that has a nested callback function, *whatslid*. Since the *whatslid* function is nested, the variable scope is such that the variables *uilab* and *uislid* can be used within the callback function without passing them. The callback function has two input arguments representing the source of the callback (in this case *uislid*) and the *eventdata*, just like GUI callback functions.

```
function uislideruilabel

    uif= uifigure;

    uif.Name = 'UI Fig Slider with Callback' ;

    uislid = uislider (uif) ;

    uilab = uilabel (uif , ' Position ', [150 200 30 15] );

    uilab.Text = 'O' ;

    uislid.ValueChangedFcn= @whatslid;

    function whatslid (source, event)

        uilab.Text = num2str(round(uislid.Value));
```

```
    end

end
```

The list of UI functions that create object components can be found in the App Designer documentation. The documentation for each function describes the properties that can be used for each, including all of the possible callback functions.