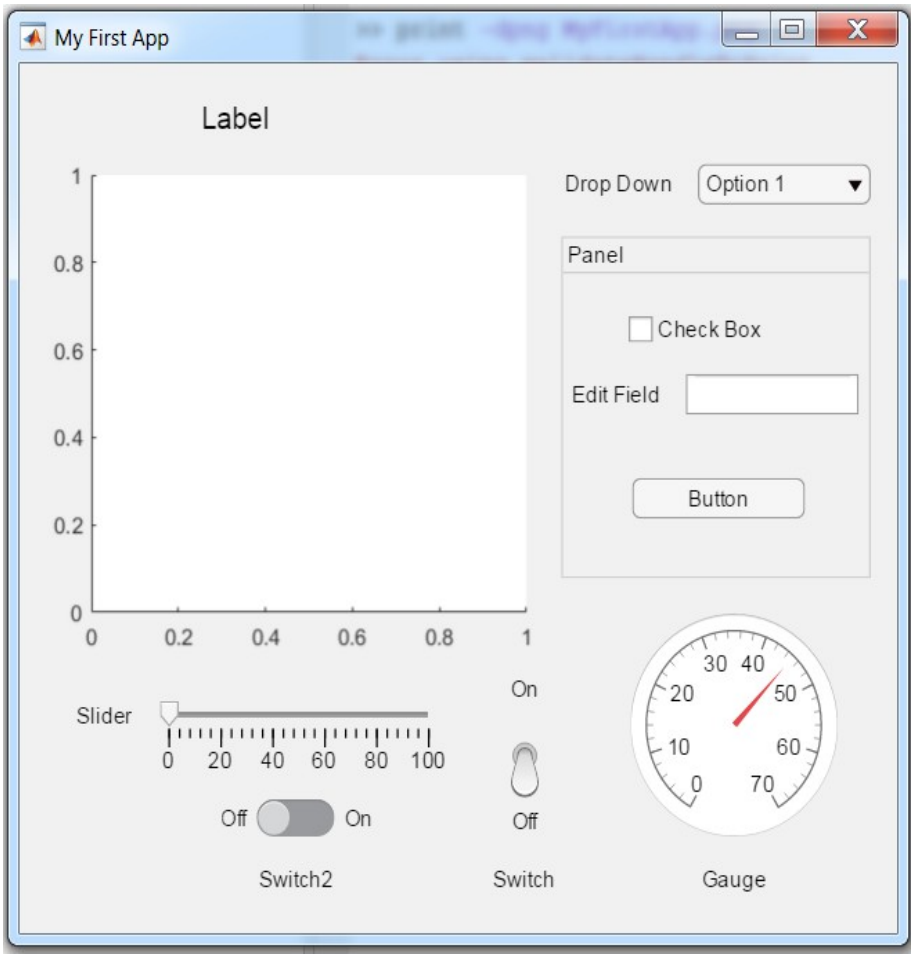
Introduction To MATLAB Interactive Graphics



Eric Peasley, Department of Engineering Science, University of Oxford
version 4.3, 2018

An Introduction to MATLAB Interactive Graphics

Table of Contents

Version 4.3

Data Types

Characters and Strings

There are two ways of storing text into a MATLAB variable. For many applications you can use either type. The main difference between the two is how they behave when you put then into an array.

Character Arrays

A character variable holds a single text character. To create a character variable, put the character between single quotation marks.

```
c = 'a'
```

A character array is an array of character variables.

```
c1 =  ['H' 'e' 'l' 'l' 'o']
```

This is not a very convenient way to create the array. Normally you use the following :-

```
c1 = 'Hello'
```

However, it is still an array, so you can do the same types of things that you do to a numerical array.

```
c2 = 'World'
ca = [c1 ' ' c2]            % Join 3 char arrays together
ca(4)                       % The 4th char in the array
ca([4 7])                   % The 4th and 7th char in the array
ca( 4:7 )                   % The 4th to 7th char in the array
ca == 'o'                   % logic array, 1 when letter 'o'
az = 'a':'z'                % All letters from a to z
length(az)                  % The number of letters in alphabet
```

String Arrays

String Arrays are arrays of text rather than arrays of single characters.

```
sa = [ "one" "two" ; "three" "four" ]
```

A single string such as `"Hello World"` is not an array, it is a single entity. You cannot use indexing on a string to extract single characters. Instead you use indexing to extract a the whole string of text.

```
sa(1,2)       % Is the string "two"
```

Other than that, strings and character arrays are almost interchangeable. Many of the functions that work with strings will also work with character arrays.

Cell Arrays

A cell array can be used to store text, numbers and and other types.

```
C = { 1 2 'Text' [1 2 3 4 5]}
C{1}                          % Extract the first number
C{3}                          % Extract the text
C{4}                          % Extract the vector
C{4}(3)                       % Extract the 3rd element from vector
```

Notice that cell arrays use curly brackets instead of square and round brackets.

Text Comparison

The function **strcmp** compares two strings

```
strcmp(ca,'hello world')      %Compare two strings
```

The result is either true or false. The case above will be false because of the capital letters in the original string. There is a case insensitive version that will give true.

```
strcmpi(ca,'hello world')     %Case insensitive version
```

The function can be used as the conditional part of a while loop or if statement.

There are several other functions that can be used to compare two strings. To find out more, enter

```
doc compare text
```

in the MATLAB command window.

Converting Numbers into Strings

There are several functions to convert numbers into strings. The most basic one is num2str.

```
S = num2str(pi)           % converts pi into a string
S = num2str(pi,'%0.1f')   % fixed point to one decimal place
S = num2str(pi,'%0.2e')   % exponential notation, 2 decimal place
S = num2str(11,'%x')      % present as a hexadecimal number
```

Anybody that has used the C programming language will recognize the format strings as being the same as used by the fprinf and sprintf functions. These functions are also available within MATLAB.

```
S = sprintf('hexadecimal %x \nand fixed point %0.1f ',11,pi)
```

`%x`	A hexadecimal number should be inserted here.
`\n`	Start a new line of text.
`%0.1f`	Insert a fixed point number with just one decimal place.

The two end arguments contain the numbers to be inserted. So this produces the following character array.

S =

'hexadecimal b

and fixed point 3.1 '

If you use double quotation marks in sprintf, it produces a string. You will find much more information about formatting with sprintf in the MATLAB documentation.

```
doc sprintf
```

Converting Strings into Numbers

There is also a function for converting a string into a number.

```
N = str2double('3.142')        %Convert a char array into a number
```

Evaluate a string as MATLAB code

You can also use a string as a command to MATLAB

```
c1 = 'sqrt(9)'          % Char array containing the MATLAB statement
A = eval(c1)            % execute the MATLAB statement in s1
```

Structures

A structure is a way of grouping different types of information into a single variable.
For example, the following creates a structure called ***Chem***.

```
Chem.Name = 'Sodium';
Chem.Symbol = 'Na';
Chem.Atomic = 11 ;
```

The variable ***Chem*** can be used on its own, to assign the structure to a new variable or to pass the entire structure to a function.

```
NewStructure = Chem ;
myfunction(Chem);
```

This makes it easier to pass vast amounts of data to a function.

Each item in a structure is called a field. To access an individual field you use the name of the structure and the name of the field separated by a full stop. You can add a new field to a structure whenever you like. The following adds a new field called ***Weight*** the the structure ***Chem***.

```
Chem.Weight = 22.99 ;
```

The same notation is used to extract the value of a field.

```
W = Chem.Weight;
```

A field can contain a number, a string, or an array.

```
Chem.Shells = [ 2 2 6 1];
shells = Chem.Shells ;
shell3 = Chem.Shells(3);
```

A field can also be a structure.

```
Person.Name = 'Sir Humphry Davy';
Person.Nationality = 'British' ;
Person.Birth = 1778;
Person.Death = 1829;
Chem.Discoverer = Person ;
Where = Chem.Discoverer.Nationality
```

This can go on for many levels. You can also have arrays of structures. This gives you an infinite variety of ways of organising your data.

Flow Control

Switch

A switch statement uses the value of a variable to select which code to execute.
For example

```
switch(DayNumber)
    case 1
       Day = 'Monday';
    case 2
       Day = 'Tuesday';
    case 3
       Day = 'Wednesday';
    case 4
       Day = 'Thursday';
    case 5
       Day = 'Friday';
    case 6
       Day = 'Saturday';
    case 7
       Day = 'Sunday';
    otherwise
        Day = [];
        errordlg('Invalid day number')
end
```

If DayNumber is 1, Day is set to Monday, if DayNumber is 2, Day is set to Tuesday etc.
If Daynumber is not in the range 1 to 7, then the statements after **otherwise** are executed.
You can put several lines of code for each case if required. You can also execute the same code for several different numbers. For example

```
switch(DayNumber)
    case{1,2,3,4,5}
        DayType = 'Week Day';
    case{6,7}
        DayType = 'Weekend';
end
```

Try Catch

Try and catch are used for error recovery. Interactive programs can generate errors because the user makes a mistake. The aim is to catch those errors and do something sensible, rather than crash the program. You place the vulnerable code in the try part and then the catch part will only execute if an error occurs in the try part. For example

```
x = linspace(-1,1,100);                    % Generate x at 100 points
try
    A = inputdlg('Enter an expression: '); %Ask for an expression
    y = eval(A{1});                        %Evaluate expression
    plot(x,y)
catch err
    errordlg(err.message)                  %Error dialogue box
end
```

In the **try** part the program tries to plot an expression entered by the user. It is quite possible that the user will enter an invalid expression and cause an error. If an error does occur, the the program will immediately jump to the **catch** part. The variable called **err**, is a structure containing information about the error that has occurred. A field called **message** in this structure contains the error message. This is displayed in an error dialogue box.

Global Variables

When you write a MATLAB function, any variables within the function are local to that function. This means that they can only be used within the function itself and cannot be accessed in any other function or the command window. Global variables can be accessed in several functions and the command window. To make a variable global, you declare that it is global at the top of your function.

```
global a b c
```

You must declare a variable as global in every function where you want to use it. This also applies to the command window. You will not be able to see any global variables within the command window until you declare the variable as global within the command window. Any declaration of global variables within a function must be at the beginning of the function.

Predefined Dialogue Boxes

Often when you are developing a Graphical User Interface, you want to bring up a small window to display a message, ask for some input or the name of a file. You could write your own app to do this. However, MATLAB includes many different types of dialogue boxes ready for you to use. To see how these work, try the following.

```
msgbox('Hello World')
msgbox('Hello World','My Title')

a = questdlg('Are you happy')
```

There are many other types of predefine dialogue boxes. To see the full list, look in the MATLAB documentation.

```
doc Dialog Boxes
```

You will use several different predefine dialogue boxes in the exercises.

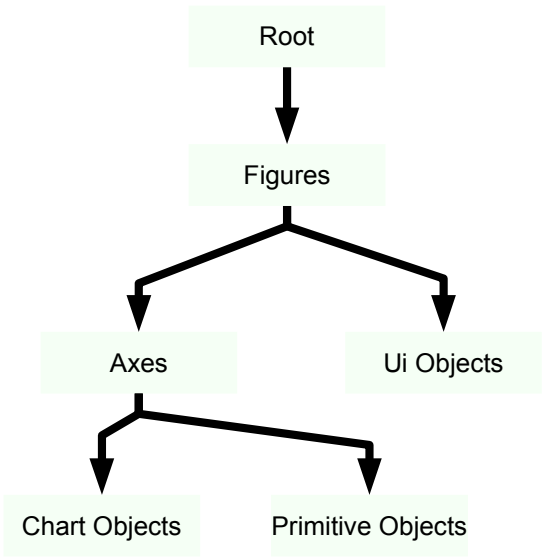
Graphics Objects

Sometimes you need to produce graphics that are different or more complex than the graphs obtained by the standard plot functions. When you use the plot function in MATLAB, the graphics produced are composed of simpler graphic objects. You can compose your own graphics using these objects in what ever way you wish.

Every graphical entity has a extensive selection of properties that can be configured to your own requirements. To access these properties you use an "object". You change an object like you change a structure, it has a field for each property. When you change a field, you change the appearance or behaviour of the graphical object.

Types of Object.

There is a hierarchy of graphical objects within MATLAB.



Root
At the top is the root, which is not really a graphics object at all. It is just a starting point for everything else. The root also has properties that are inherited by its children below.
The function **groot** returns the root object.

Figures
Figures are the windows that contain the graphics.

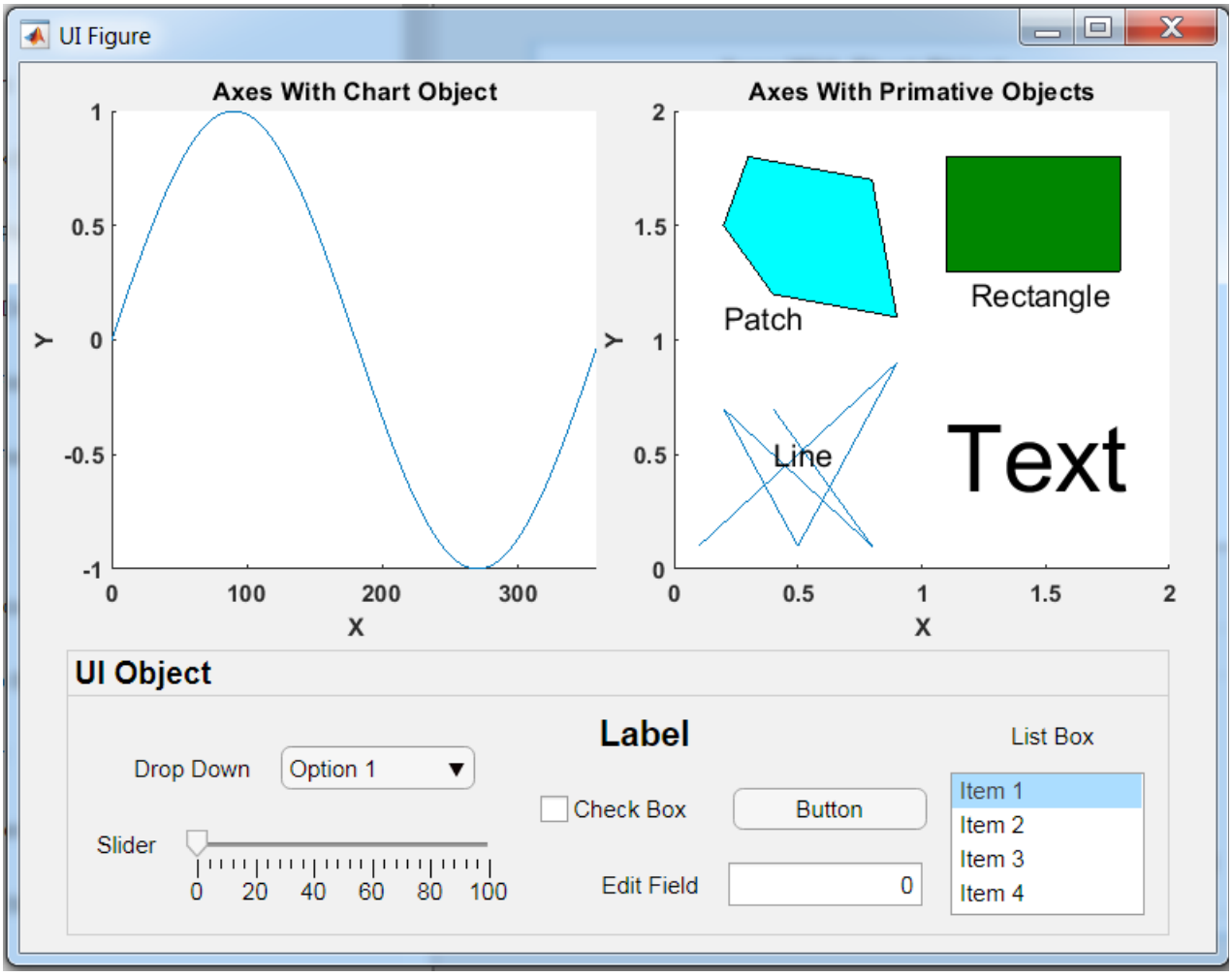
Axes
An axes is like the paper that a graph is plotted onto. There can be several axes in a figure. Each axes in a figure is a child of the figure and the figure is a parent of the axes.

UI Objects
The other type of object that you can put directly into a figure are the User Interface (UI) objects such as push buttons, sliders and check boxes.

Chart Objects
Chart objects are the 2D graphs produced by plot, bar, semilog and the 3D graphs produced by surf, mesh and plot3 etc.

Primitive Objects
Then there are the low level graphical objects such as lines, text, rectangles and patches. A patch is a *n* sided shape, defined by the coordinates of the corners. These are all children of an axes.

Example of a MATLAB figure



Object Creation

There is a function for each type of object that will create that particular object.
For example

```
hf = figure
```

Will create a figure and the object ***hf*** can be use later to access the properties of the figure. At the same time as you create a figure, you can set object properties.

```
hf = figure('Position',[100,100,500,500])
```

Which will create a 500 by 500 pixel figure, 100 pixels from the bottom and left of the screen. Although there are many different object creation functions, they are all similar in the way that you use them. In the general form of the function, you specify the **name** of the property and then the **value** you want to assign to the property, this is known as name-value pairs. You can put in as many name-value pairs as you like.

```
hf = figure('Position',[100,100,500,500],  ...
            'Color',[.2,.8,.8],  ...
            'Resize','off')
```

MATLAB is an American program, so you need to use “Color”, the American spelling. If you change more than two properties, the statements lines can get very long. Above I have split the statement over several lines using ellipses(...), so that there is one property per line.

You use the object ***hf*** to access the properties of the object like you access the fields of a structure. An alternative to the above

```
hf = figure;
hf.Position = [100,100,500,500];
hf.Color = [.2,.8,.8];
hf.Resize ='off';
```

Object Properties

A figure has over sixty different properties. Other objects have a similar amount. So it is fairly unlikely that you will remember the names of all the properties and how to use them. The properties of each type of object are listed in the MATLAB online documentation. When you display a graphical object in the command window, there is a link to the documentation. Enter the name of the object and click on the blue, underlined link.

```
>> hf

hf =

  Figure (1) with properties:

      Number: 1
        Name: ''
```

Current Figure

When you create an axes, it is put into the current figure. MATLAB keeps the record of which figure is the current figure. If you create a new figure, that becomes the current figure. You can change the current figure using the figure function.

```
figure(hf)        % Change current figure to hf
ha = axes;        % Create a new axes in hf called ha.
```

You can also find the “object” of the current figure with the the function gcf.

```
hcf = gcf;        % Get current figure
```

If no figure exists when you create an axes, a figure will automatically be created and become the current figure. If you click on a figure, then that becomes the current figure.

Current Axes

The current axes is similar to the current figure. If you create an object that needs to be within an axes, it is put into the current axes. The current axes is the last axes created or you can switch the current axes using the axes function.

```
axes(ha)      % Change current axes to ha
gca           % Get the current axes
```

You can also select a different current axes by clicking on an axes.

Specified Destination

You can also specify which figure or axes the object will be in when you create the object.

```
hf = figure;              % Create a figure
ha= axes(hf);             % Create and axes in the figure hf
x = 0:0.1:1;
y = x.^2;

hp = plot(ha,x,y);        % Plot the graph hp in the axes ha
```

Simplified Calling Syntax

To make things easier to use, some functions have a simplified form. For example, to write the text “Hello world” on an axes at positions (0.5, 0.5), I could use

```
text('Position',[0.5 0.5],'String','Hello')
```

However, in practice you will always want to specify the string and the position. So there is a simper way of doing this.

```
text(0.5,0.5,'Hello There')
```

Other functions have to be entered in a simpler form. For example, you would not want to enter a plot function in this form.

```
plot('XData',x,'YData',y,'Color','r')    % does not work
```

However, the plot functions does create a graphical object and you can interact with its properties just like any other graphical object.

```
hp = plot(x,y,'r','LineWidth',2)
hp.LineStyle = '-.'
xd = hp.XData;
```

Changing Object Properties

There are a number of ways to change the properties after it has been created. The original method used the function **get** to find a properties value and the function **set** to change the value.

```
lw = get(hp,'LineWidth') %Read line width of plot hp
set(hp,'LineWidth',3)    %Set line width of plot hp
```

Since MATLAB release R2014b, you no longer need to use the set and get functions to change object properties. The modern equivalent to the above is

```
lw = hp.LineWidth
hp.LineWidth = 3
```

Here are more examples

```
figcolour = hf.Color          %Read the colour of figure hf
hf.Color = 'w'                %Set the colour of figure hf to white
```

Unfortunately, **gcf** is a function and not an object. You can use **gcf** in **get** and **set** but you cannot use gcf with the dot notation.

```
get(gcf,'Color')          %Display colour of the current figure
get(gcf)                  %Display all proprieties of the figure
set(gcf,'Color','g')      %Set the colour of the current figure
```

Other Object Graphics Functions

```
isgraphics(h)                 %Test if h is a graphics object
```

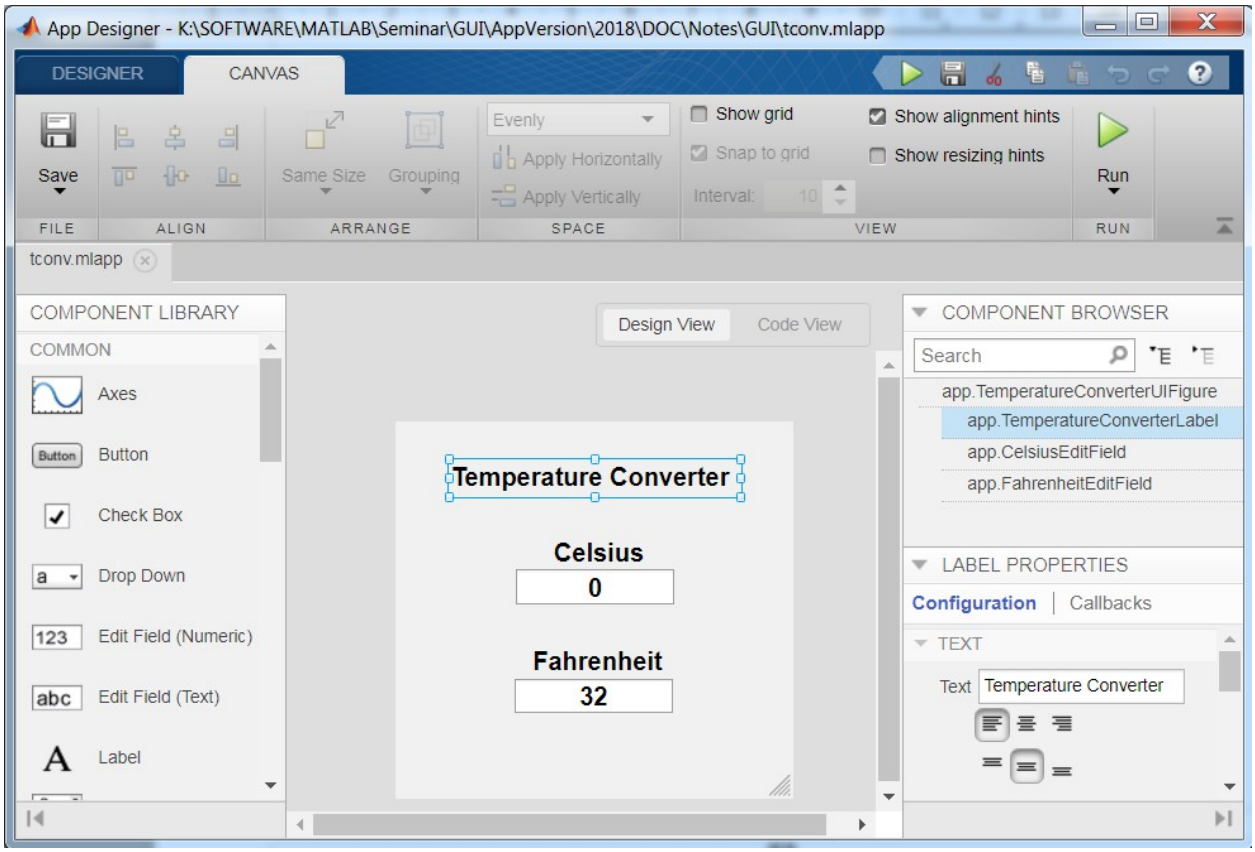
Returns true if **h** is the handle of a graphical object. Can be used in the condition part of if and while.

```
cla                   %clear current axes
cla(ha)               %clear axes with object ha
clf                   %clear current figure
clf(hf)               %clear figure hf

delete(h)             %Delete object h
```

App Designer

The **App Designer** is a Computer Aided Design tool for MATLAB apps. It allows you to select different types of graphic objects and drag them into position on to a figure. The example below show an **App Designer** window containing the design for a temperature conversion app.



This shows the **Design View**. This app uses two types of object. The white boxes are **Edit Field** objects. Users can type into an **Edit Field** box. Text at the top of the App is a **Label**. Each of the **Edit Fields** automatically has a built in **Label**. You can select an object, either in the **Component Browser** or by clicking on the object. Then you can change its properties.

App Designed Code

The App Designer automatically generates the code to create the app. It prevents you for editing the code that it adds. You can produce your own apps without understanding what is in this code, so feel free to skip the rest of this page.

The general form of the code generated is the same for every app.

Properties
At the top of the code is the properties. This is a list of things that are accessible inside the app. To start with, the list consists of all the graphical objects that make up the app. At the top of the list will be the figure itself. This is what it looks like in the temperature converter app.

```
TemperatureConverterUIFigure  matlab.ui.Figure
```

This defines the name of the property and what kind object it is. In this case it is a `matlab.ui.Figure` object. This is part of the app object. To use this property in the rest of code you use:

```
app.TemperatureConverterUIFigure
```

The Create Components Function
The function **CreateComponents** contains all the code to produce all the graphical objects specified in the **Design View** of the App Designer. At the top, the figure is created. This is what it looks like in the temperate converter app.

```
% Create TemperatureConverterUIFigure
app.TemperatureConverterUIFigure = uifigure;
app.TemperatureConverterUIFigure.Position = [100 100 251 239];
app.TemperatureConverterUIFigure.Name = 'Temperature Converter';
```

Below this will be all the code to create the rest of the objects.

The Main App Function
Below **CreateComponents** is the main app function. This is what executes when you run the app. It does two things: Calls **CreateComponents** and registers the app.

The Delete Function
The last function is the **delete** function which tidies things up when the app is deleted.

Events and Callback Functions

To make it possible to interact with the graphics, many of the graphical objects look for particular events to occur. You can configure an object to run a **Callback Function** when an event happens.

When first created, the temperature converter app does nothing but create the graphical objects. To make it work, callback functions needed to be added. The App Designer will do most of the work for you. You can tell it that you want a **Callback Function** for a particular event on a particular object . Then all you have to do is add the code into the **Callback Function** to do what ever it is you want to do.

Take for example the Celsius **Edit Field** on the temperature converter. When somebody enters a temperature into the **Edit Field**, we want the app to display the temperature in Fahrenheit in the other **Edit Field**. So we get the **App Designer** to create a **Value Changed Callback Function** and we add the following code into the function:

```
C = app.CelsiusEditField.Value;  % Get the number from Edit Field
F = C*9/5+32;                      % Convert to Fahrenheit
app.FahrenheitEditField.Value = F;   % Write F to Edit Field
```

Then we add a **Value Changed Callback Function** for the Fahrenheit **Edit Field**.

```
F = app.FahrenheitEditField.Value;  % Get number from Edit Field
C = (F-32)*5/9;                      % Convert to Celsius
app.CelsiusEditField.Value = C;      % Write C to Edit Field
```

That is all you need to do to get the app working.