Graphical User Interface with MATLAB App Designer

PN130 | Posted on March 25, 2021 | Updated on August 4, 2025

Jessy ANÇAY
Sales & Project Engineer
imperix • in

Table of Contents

This note provides step-by-step guidance to implement a basic Graphical User Interface (GUI) with MATLAB app designer. Throughout the page, a straightforward application is put together. It incorporates the main elements required for a GUI such as UI components, callback functions, timers, and background tasks.

Basic concepts of App Designer are introduced here. It is also possible to use a GUI made with App Designer as an OPC UA clients. Further details on the subject are found on the page: OPC UA client with the Industrial Communication Toolbox. For a more concrete application example applied to a real Imperix converter, please refer to the page Custom user interface to operate Imperix converters, which dives deeper into the practical aspects related to the development of a GUI to remotely control the B-Box RCP prototyping controller.

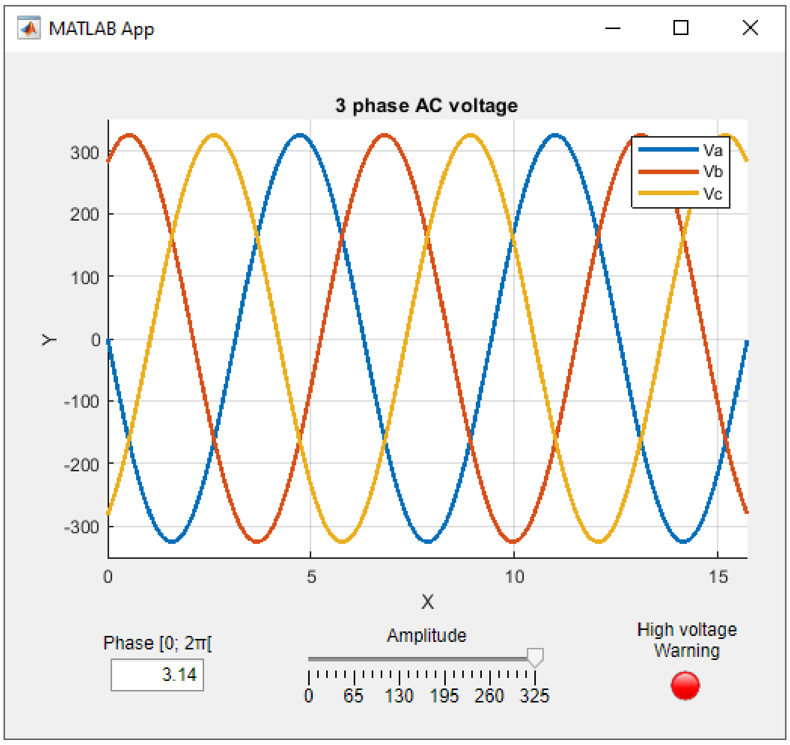


Figure 1: **Basic example of a GUI with App Designer**

Software resource

The code of the application (MATLAB version R2022a or newer required) used as an example throughout this page can be downloaded using the link below. This basic GUI simply plots three-phase sinusoidal voltage whose amplitude and phase can respectively be adjusted using a slider and a text input.

Download **ExampleApp.zip**

Making a GUI with App Designer

Introduced in MATLAB version R2016a and part of the MATLAB basic package, App Designer allows to conveniently design graphical user interfaces by dragging and dropping visual components. Actions and processes are meant to be implemented in the well-known MATLAB programming language.

As mentioned, no add-ons or toolboxes are required. However, at a later stage, the MATLAB compiler add-on can be used to share MATLAB programs as standalone applications. Applications can then be launched on any computer using the free MATLAB Runtime libraries (a standalone set of shared libraries, MATLAB code, and other files that enables the execution of MATLAB files on computers without an installed version of MATLAB).

MATLAB Help Center provides many tutorials on their App Designer product, e.g. Create and Run a Simple App Using App Designer.

To introduce the main concepts, the example from Figure 1 will be recreated from scratch throughout this page. This example plots three sinewaves whose amplitude and phase can be modified using a slider and an edit field, respectively. Also, the GUI incorporates a lamp that changes color according to the amplitude of the sinewaves.

Implementation principles in App Designer

To start App Designer from MATLAB, type `appdesigner` in the Command Window or, starting in version R2019b, App Designer can be opened by clicking the Design App button in the Apps tab.

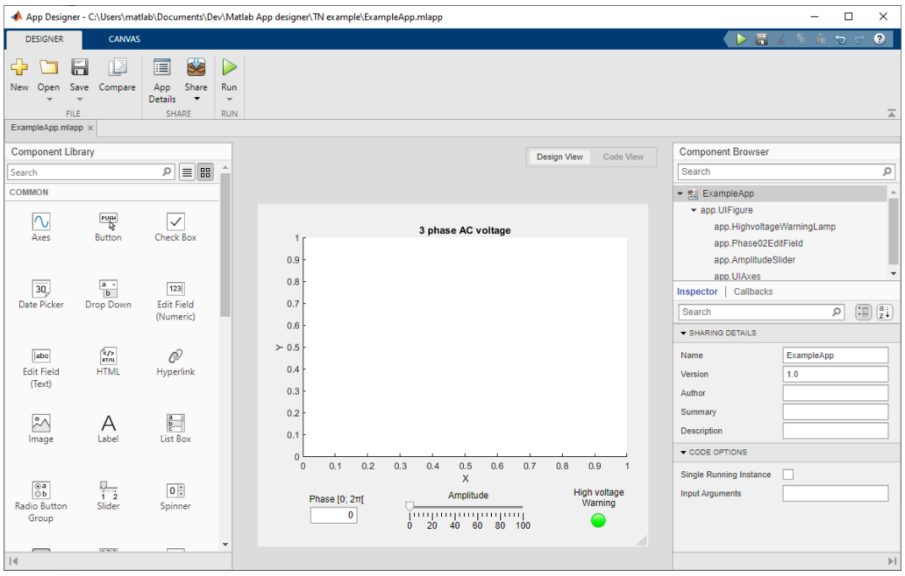
Creating UI components

Upon opening a blank project in MATLAB App Designer, the first step is to add UI components. To recreate the example from Figure 1:

- Add the widgets: *Axes*, *Edit Field (Numeric)*, *Slider,* and *Lamp* from the component library
- Rename their labels to match the screenshot below

Note that App Designer will then automatically add these widgets to the Component Browser and name them according to their labels.

At this point, the *Design View* should look similar to the screenshot from Figure 2.



Name: ExampleApp
Version: 1.0
Author:
Summary:
Description:
Single Running Instance: ☐
Input Arguments:

Figure 2: **Preview of the GUI in App Designer**

Share data within a GUI in App Designer

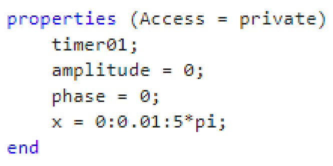
When designing a GUI with App Designer, it is often useful to be able to access variables from multiple callbacks or functions. This can be done using properties as they are accessible from anywhere inside the application.

To declare properties proceed as follows:

- Under the *Code View*, in the *Editor* tab, click on *Property*. The code to declare a property is automatically added.
- Add the code below inside the properties section to declare the required variables for the example.

```
timer01;
amplitude = 0;
phase = 0;
x = 0:0.01:5*pi;
```
Code language: Matlab (matlab)

Your code should look somewhat like the screenshot below.



```
properties (Access = private)
    timer01;
    amplitude = 0;
    phase = 0;
    x = 0:0.01:5*pi;
end
```

The `timer01` property will later be used to instantiate a timer while the three other properties are used to store the sinewaves' parameters. To then get or set a property in your code, use the dot notation: `app.myProperty`.

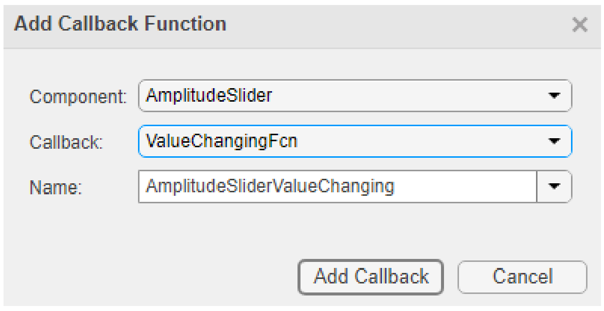
Defining callback functions

The next step is to add callbacks to the action widgets. Callbacks basically contain the code that will be executed when the user interacts with the corresponding widget.

To do so:

- Switch to the *Code View* in App Designer and, under the *Editor* tab, click on *Callback*. A pop-up window like the one shown in Figure 3 should open.
- In this pop-up window, select the *AmplitudeSlider* component and the *ValueChangingFcn* callback and click on *Add Callback*. MATLAB App Designer will automatically add the corresponding callback function in the code.
- Add the following code inside the newly created function to update the *Axes* UI scaling and the `amplitude` property.

```
%Update the amplitude
app.UIAxes.XLim = [-0 5*pi];
app.UIAxes.YLim = [-350 350];
app.amplitude = event.Value;
```
Code language: Matlab (matlab)


Add Callback Function

Component: AmplitudeSlider

Callback: ValueChangingFcn

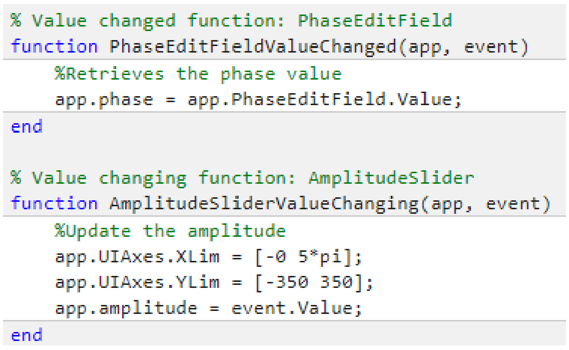
Name: AmplitudeSliderValueChanging

Figure3: **Add callback functions to widgets**

- As done before, add a callback to the *Phase EditField* UI but this time select the *ValueChangedFcn* callback.
- Enter the code below to the slider's callback function to store the *Phase EditField*'s value:

```
%Retrieves the phase value
app.phase = app.PhaseEditField.Value;
```
Code language: Matlab (matlab)

Note that the components and their corresponding callback functions (seen in Figure 3) are automatically named by App Designer according to the user-defined components' label. The code for these two callbacks should look like the screenshot below.


```
% Value changed function: PhaseEditField
function PhaseEditFieldValueChanged(app, event)
    %Retrieves the phase value
    app.phase = app.PhaseEditField.Value;
end

% Value changing function: AmplitudeSlider
function AmplitudeSliderValueChanging(app, event)
    %Update the amplitude
    app.UIAxes.XLim = [-0 5*pi];
    app.UIAxes.YLim = [-350 350];
    app.amplitude = event.Value;
end
```

Timer implementation for background tasks

Implementing timers allows for performing background tasks that can be repeated periodically. It can be used to update variables' values or in this case to refresh the function plotted inside the *Axes* UI. The first step to implement a timer is to create a startup function.

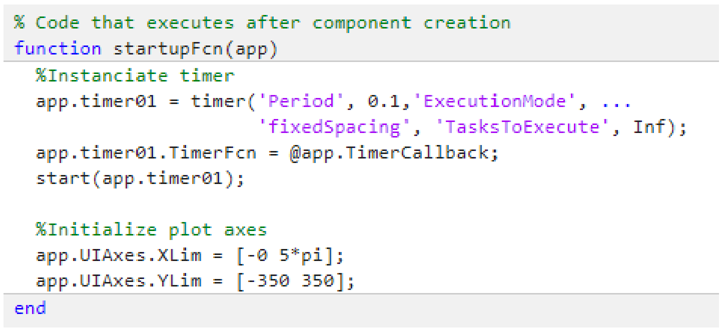
Timer declaration

Timers are usually declared in the startup function when developing a GUI in App Designer. A startup function is a specific function that is executed when the application is first launched before the user can even interact with the GUI. This is where the timer will be configured and started.

To do so:

- Right-click on the app's name (top node) in the Component Browser, hover Callbacks, and select Add StartupFcn callback. The code for the startup function will be added to the Code View.
- Add the code below inside the newly created function. It declares a timer with a period of 0.1 seconds, assigns a callback function to it, and starts the timer.

```
%Instanciate timer
app.timer01 = timer('Period', 0.1,'ExecutionMode', ...
                    'fixedSpacing', 'TasksToExecute', Inf);
app.timer01.TimerFcn = @app.TimerCallback;
start(app.timer01);
```
Code language: Matlab (matlab)



```
% Code that executes after component creation
function startupFcn(app)
  %Instanciate timer
  app.timer01 = timer('Period', 0.1,'ExecutionMode', ...
                      'fixedSpacing', 'TasksToExecute', Inf);
  app.timer01.TimerFcn = @app.TimerCallback;
  start(app.timer01);

  %Initialize plot axes
  app.UIAxes.XLim = [-0 5*pi];
  app.UIAxes.YLim = [-350 350];
end
```

Timer callback/background task definition

Timer callbacks are functions that are executed at a frequency defined by the timer's period. In this specific case, the timer callback can be seen as a background task, used to update the three sinewaves plotted in the *Axes* UI.

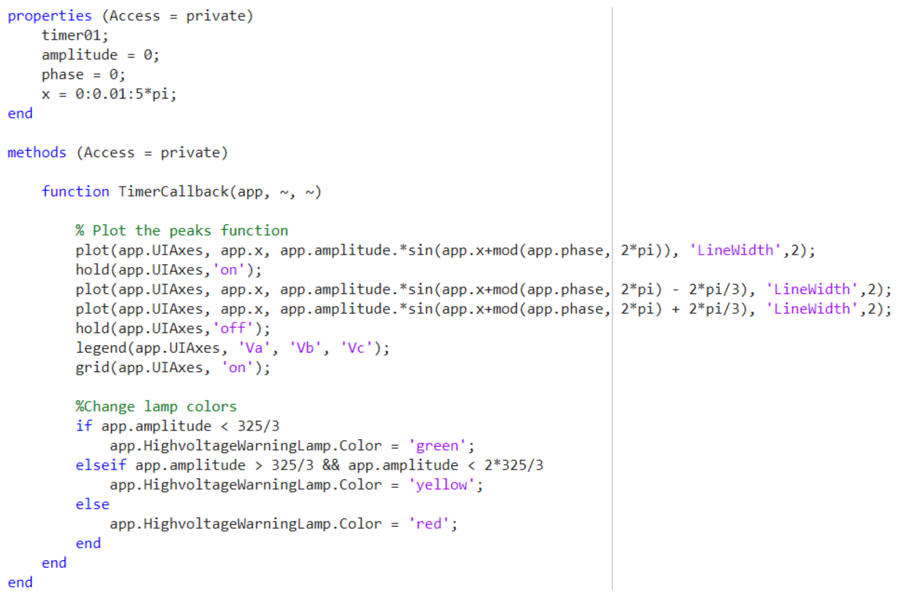
To define the timer callback, proceed as follows:

- Under the *Code View*, in the *Editor* tab, click on *Function*
- Replace the function code with the code below. This code plots three sinewaves, multiplied by the amplitude defined by the *Slider* UI and phase shifted by the value entered in the *EditField* UI.

```
function TimerCallback(app, ~, ~)

    % Plot three sinewave functions
    plot(app.UIAxes, app.x, app.amplitude.*sin(app.x+mod(app.phase, 2*pi)), 'LineWidth',2);
    hold(app.UIAxes,'on');
    plot(app.UIAxes, app.x, app.amplitude.*sin(app.x+mod(app.phase, 2*pi) - 2*pi/3), 'LineWidth',2);
    plot(app.UIAxes, app.x, app.amplitude.*sin(app.x+mod(app.phase, 2*pi) + 2*pi/3), 'LineWidth',2);
    hold(app.UIAxes,'off');
    legend(app.UIAxes, 'Va', 'Vb', 'Vc');
    grid(app.UIAxes, 'on');

    %Change lamp colors
    if app.amplitude < 325/3
        app.HighvoltageWarningLamp.Color = 'green';
    elseif app.amplitude > 325/3 && app.amplitude < 2*325/3
        app.HighvoltageWarningLamp.Color = 'yellow';
    else
        app.HighvoltageWarningLamp.Color = 'red';
    end
end
```
Code language: Matlab (matlab)



```
properties (Access = private)
    timer01;
    amplitude = 0;
    phase = 0;
    x = 0:0.01:5*pi;
end

methods (Access = private)

    function TimerCallback(app, ~, ~)

        % Plot the peaks function
        plot(app.UIAxes, app.x, app.amplitude.*sin(app.x+mod(app.phase, 2*pi)), 'LineWidth',2);
        hold(app.UIAxes,'on');
        plot(app.UIAxes, app.x, app.amplitude.*sin(app.x+mod(app.phase, 2*pi) - 2*pi/3), 'LineWidth',2);
        plot(app.UIAxes, app.x, app.amplitude.*sin(app.x+mod(app.phase, 2*pi) + 2*pi/3), 'LineWidth',2);
        hold(app.UIAxes,'off');
        legend(app.UIAxes, 'Va', 'Vb', 'Vc');
        grid(app.UIAxes, 'on');

        %Change lamp colors
        if app.amplitude < 325/3
            app.HighvoltageWarningLamp.Color = 'green';
        elseif app.amplitude > 325/3 && app.amplitude < 2*325/3
            app.HighvoltageWarningLamp.Color = 'yellow';
        else
            app.HighvoltageWarningLamp.Color = 'red';
        end
    end
end
```

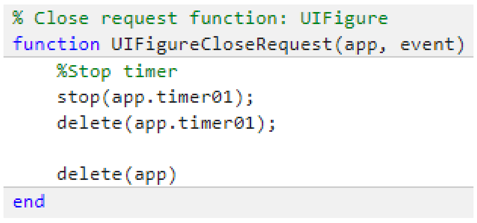
Timer shutdown

Finally, timers need to be stopped when closing the application. Similar to the startup function, the `UIFigureCloseRequest` function is called when the application is closed and can therefore be used to stop timers.

To do so:

- Right click on the UI Figure (second node) in the *Component Browser*, hover *Callbacks,* and select *Add UIFigureCloseRequest callback*. The function's code will be added to the *Code View*.
- Add the code below inside the newly created function. It stops and deletes the timer used for the background task.

```
%Stop timer
stop(app.timer01);
delete(app.timer01);Code language: C++ (cpp)
```



```
% Close request function: UIFigure
function UIFigureCloseRequest(app, event)
    %Stop timer
    stop(app.timer01);
    delete(app.timer01);

    delete(app)
end
```

Closing comments

Implementing a GUI with App Designer is a rather quick and easy task since it requires little knowledge of programming. Besides, this relies on the well-known MATLAB environment that ACG SDK users are often already familiar with. It is however important to keep in mind that one of the main drawbacks of this approach is the performance because it relies on executing an interpreted MATLAB code, which is a CPU-intensive task on the PC.

Finally, it can be attractive to use GUIs to easily interact with power converters. It can, as further discussed on the page Custom user interface to operate Imperix converters, greatly simplify the operation of more complex converters.