

MATLAB GUI Tutorial

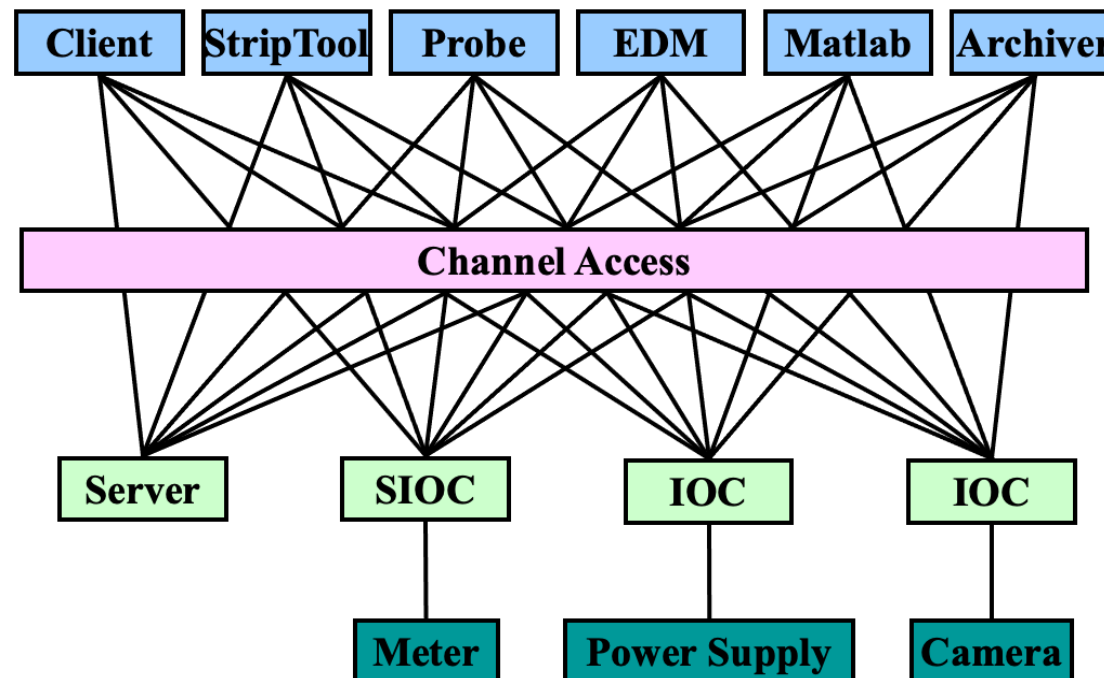
Using App Designer and Object-Oriented Programming

Sharon Perez

June 2024

Experimental Physics and Industrial Control System (EPICS)

- EPICS allows users to interact (read/write) with components of the accelerator
- Helpful links for more information:
 - EPICS website: <https://epics-controls.org/about-epics/>
 - EPICS documentation: <https://docs.epics-controls.org/en/latest/index.html>
 - US Particle Accelerator School (USPAS): https://controlssoftware.sns.ornl.gov/training/2022_USPAS/



Experimental Physics and Industrial Control System (EPICS)

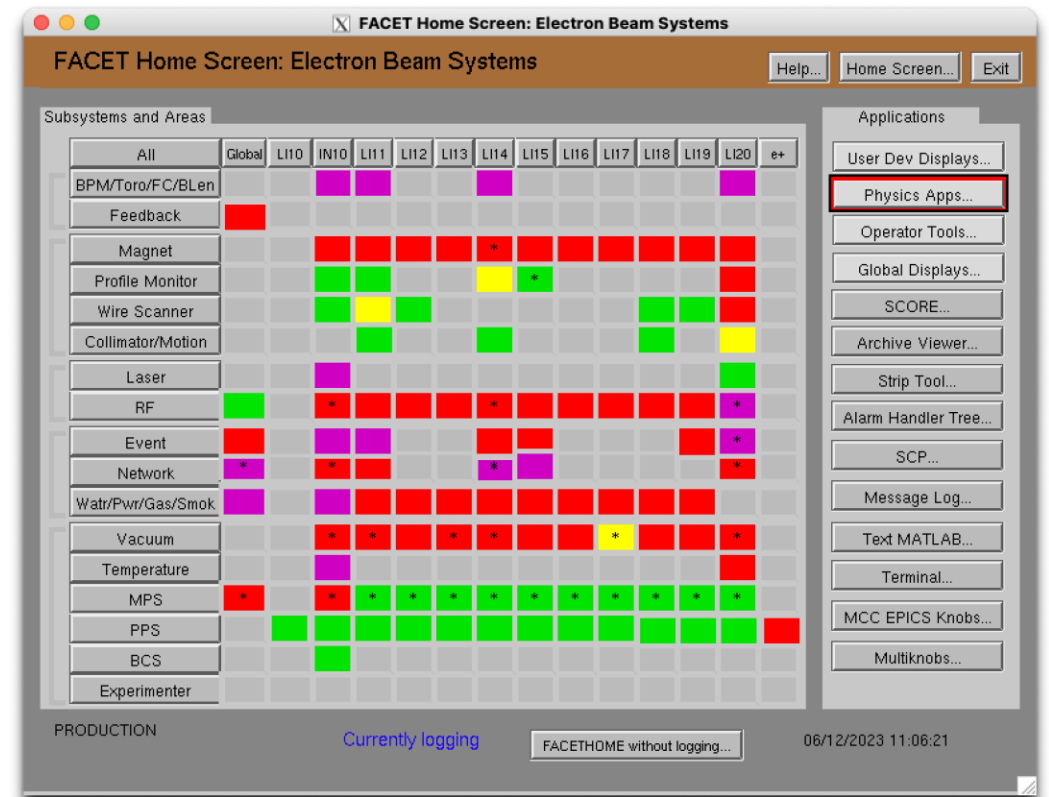
- **Process Variable (PV):** a variable associated with a component of the accelerator
 - e.g., the power status of a camera, the name of a motor
- Can “get” and “put” values in MATLAB:

```
>> lcaGet('CAMR:LI20:100:NAME')
```

ans =

1x1 cell array

{'SYAG'}
- EDM (Extensible Display Manager): tool to create displays
- Launch FACET Home Screen:
 - `>> facethome &`



Experimental Physics and Industrial Control System (EPICS)

In this tutorial, we will be plotting the following PVs:

- SIOC:SYS1:ML00:AO951
- SIOC:SYS1:ML00:AO952

FACET Home Screen > Physics Apps > Matlab
PVs ... > Matlab Support PVs ... > ML00 951-000



ML00 Matlab Support pvs.				
SIOC:SYS1:ML00:AO951	Uniform [0, 1] Random number	0.000000	egu	comment
SIOC:SYS1:ML00:AO952	Uniform [0, 1] Random number	0.000000	egu	comment
SIOC:SYS1:ML00:AO953	Normally distributed randoms	inf	egu	comment
SIOC:SYS1:ML00:AO954	TSE. used in tcav feedback	0		closes/opens loop
SIOC:SYS1:ML00:AO955	Tse setpoint in feedback	-0	mm	setpoint in xtcav fdbk
SIOC:SYS1:ML00:AO956	Test PV	1	egu	comment
SIOC:SYS1:ML00:AO957	LEM REF Valid	1	bool	F2_LEMApp.m
SIOC:SYS1:ML00:AO958	spare	0	egu	comment
SIOC:SYS1:ML00:AO959	Spectrometer Scan PV	0		scanFunc_Spect_Quad_Scan.m
SIOC:SYS1:ML00:AO960	spare	0	egu	MoveDelayStageSmartPV
SIOC:SYS1:ML00:AO961	spare	0	egu	MoveAngleSmartXPV
SIOC:SYS1:ML00:AO962	spare	0	egu	MoveAnglesmartYPV
SIOC:SYS1:ML00:AO963	corrPlot Test	0	rand	SIOC for testing corrPlot
SIOC:SYS1:ML00:AO964	spare	0	egu	comment
SIOC:SYS1:ML00:AO965	spare	0	egu	comment
SIOC:SYS1:ML00:AO966	IP2B Counts	2335	egu	comment
SIOC:SYS1:ML00:AO967	spare	0	egu	comment
SIOC:SYS1:ML00:AO968	spare	0	egu	comment
SIOC:SYS1:ML00:AO969	spare	0	egu	comment
SIOC:SYS1:ML00:AO970	Filter	2	egu	comment
SIOC:SYS1:ML00:AO971	Buffer Java	1	egu	comment
SIOC:SYS1:ML00:AO972	Filter target position	34	egu	comment
SIOC:SYS1:ML00:AO973	Target position for grating	0	egu	comment
SIOC:SYS1:ML00:AO974	TimeLapseG	2	egu	comment
SIOC:SYS1:ML00:AO975	TimeLapseF	0	egu	comment
SIOC:SYS1:ML00:AO976	spare	0	egu	comment
SIOC:SYS1:ML00:AO977	spare	0	egu	comment
SIOC:SYS1:ML00:AO978	spare	0	egu	comment
SIOC:SYS1:ML00:AO979	spare	0	egu	comment
SIOC:SYS1:ML00:AO980	spare	900	egu	aximg1 camera BM x
SIOC:SYS1:ML00:AO981	spare	180	egu	aximg1 camera BM y
SIOC:SYS1:ML00:AO982	spare	150	egu	aximg2 camera BM x
SIOC:SYS1:ML00:AO983	spare	490	egu	aximg2 camera BM y
SIOC:SYS1:ML00:AO984	spare	1140	egu	aximg3 camera BM x
SIOC:SYS1:ML00:AO985	spare	100	egu	aximg3 camera BM y

Launching MATLAB

- Open FastX
- Launch a desktop session
- Open a terminal
- `>> ssh -Y mcclogin`
- `>> ssh fphysics@facet-srv01`
- Select your profile
- `>> cd git_work/matlabTNG`
- `>> matlab2020a &`

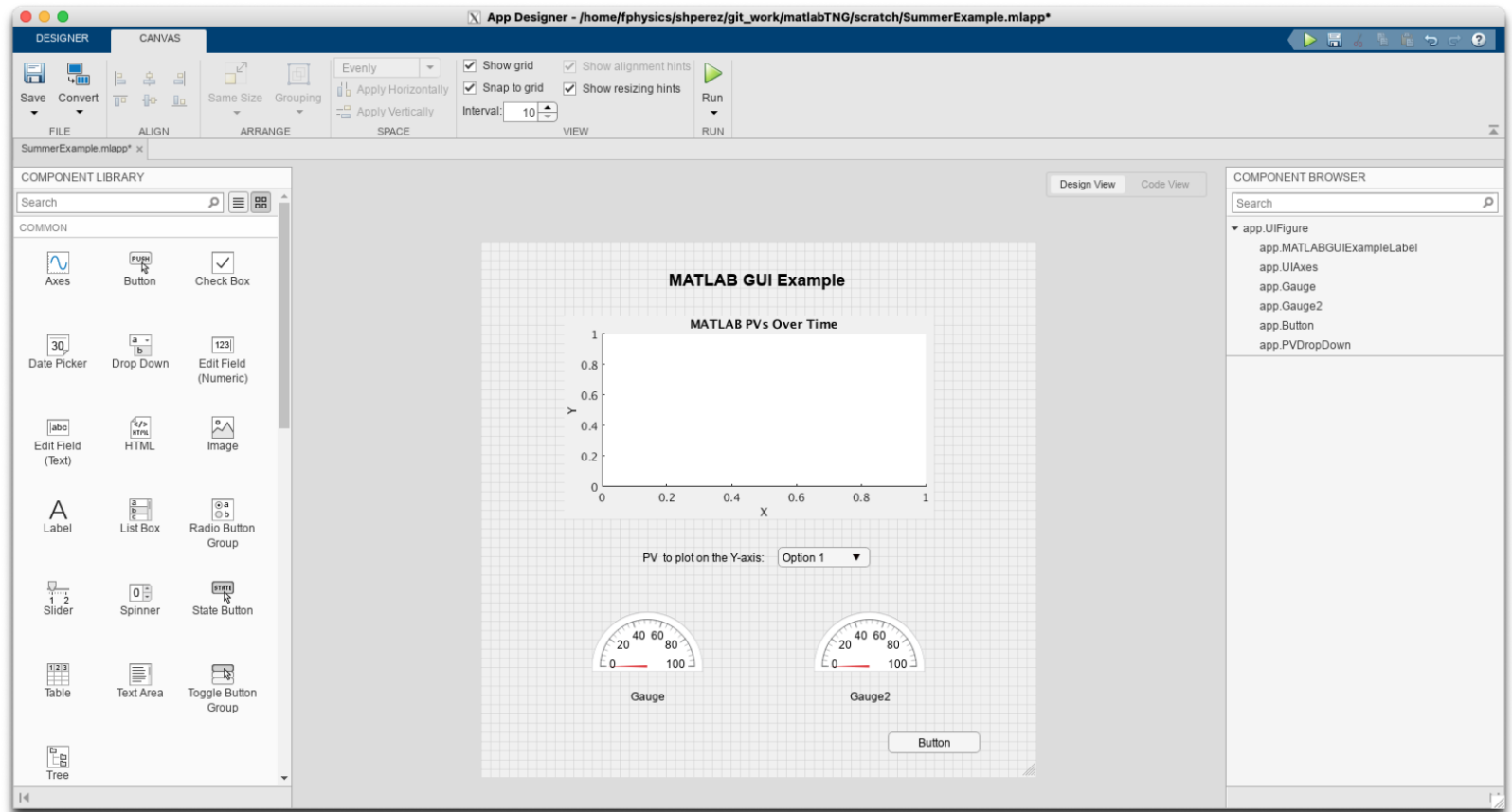
Helpful Confluence links:

<https://confluence.slac.stanford.edu/display/SCSPub/FastX>

<https://confluence.slac.stanford.edu/display/FACET/Matlab+2020>

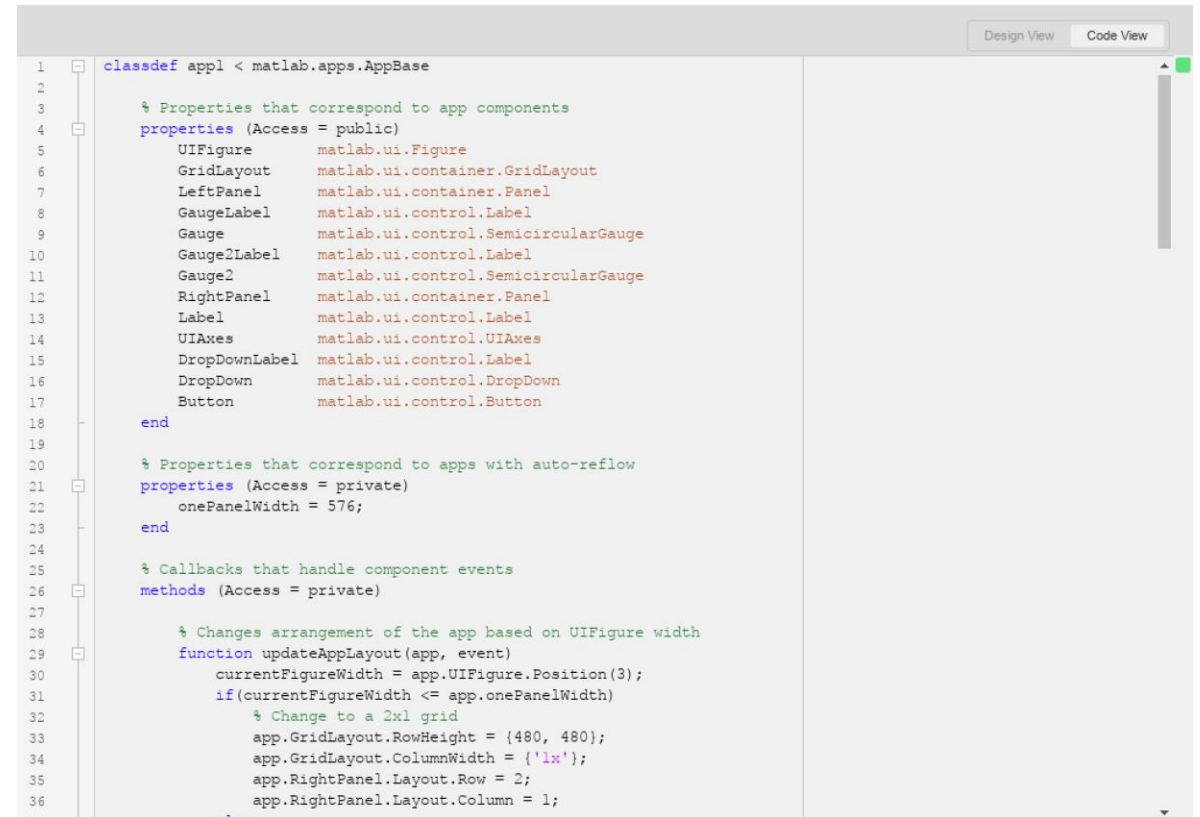
Step 1: Create app and its visual components

- Open “Examples” folder
- Start by launching App Designer:
 - `>> appdesigner`
- Create a new Blank App
- Drag and drop the following components from the Component Library on the left:
 - Label
 - Gauge (or 90 Degree Gauge, or Semicircular Gauge) (2)
 - Axes
 - Drop Down
 - Button



Step 1: Create app and its visual components

- Toggle between "Design View" and "Code View" to see the code that was autogenerated when you dragged the components onto the canvas
- *Note that the code is grayed out, which means it cannot be edited*
- Edit the app as desired. Suggested changes:
 - Change the title of the app
 - Label one of the gauges "PV1" and the other "PV2"
 - Change the limits of the gauges to [0,1]
 - Title the plot "MATLAB PVs over time"
 - Change the drop-down options to "PV1" and "PV2"
 - Edit the button text to say "Close App"



```
1 classdef appl < matlab.apps.AppBase
2
3     % Properties that correspond to app components
4     properties (Access = public)
5         UIFigure      matlab.ui.Figure
6         GridLayout    matlab.ui.container.GridLayout
7         LeftPanel     matlab.ui.container.Panel
8         GaugeLabel    matlab.ui.control.Label
9         Gauge         matlab.ui.control.SemicircularGauge
10        Gauge2Label   matlab.ui.control.Label
11        Gauge2        matlab.ui.control.SemicircularGauge
12        RightPanel    matlab.ui.container.Panel
13        Label         matlab.ui.control.Label
14        UIAxes        matlab.ui.control.UIAxes
15        DropDownLabel matlab.ui.control.Label
16        DropDown      matlab.ui.control.DropDown
17        Button        matlab.ui.control.Button
18    end
19
20     % Properties that correspond to apps with auto-reflow
21     properties (Access = private)
22         onePanelWidth = 576;
23    end
24
25     % Callbacks that handle component events
26     methods (Access = private)
27
28         % Changes arrangement of the app based on UIFigure width
29         function updateAppLayout(app, event)
30             currentFigureWidth = app.UIFigure.Position(3);
31             if(currentFigureWidth <= app.onePanelWidth)
32                 % Change to a 2x1 grid
33                 app.GridLayout.RowHeight = {480, 480};
34                 app.GridLayout.ColumnWidth = {'1x'};
35                 app.RightPanel.Layout.Row = 2;
36                 app.RightPanel.Layout.Column = 1;
37             end
38         end
39     end
40 end
```

Step 2: Create a "GUI support" class definition

Why use a GUI support class?

App Designer

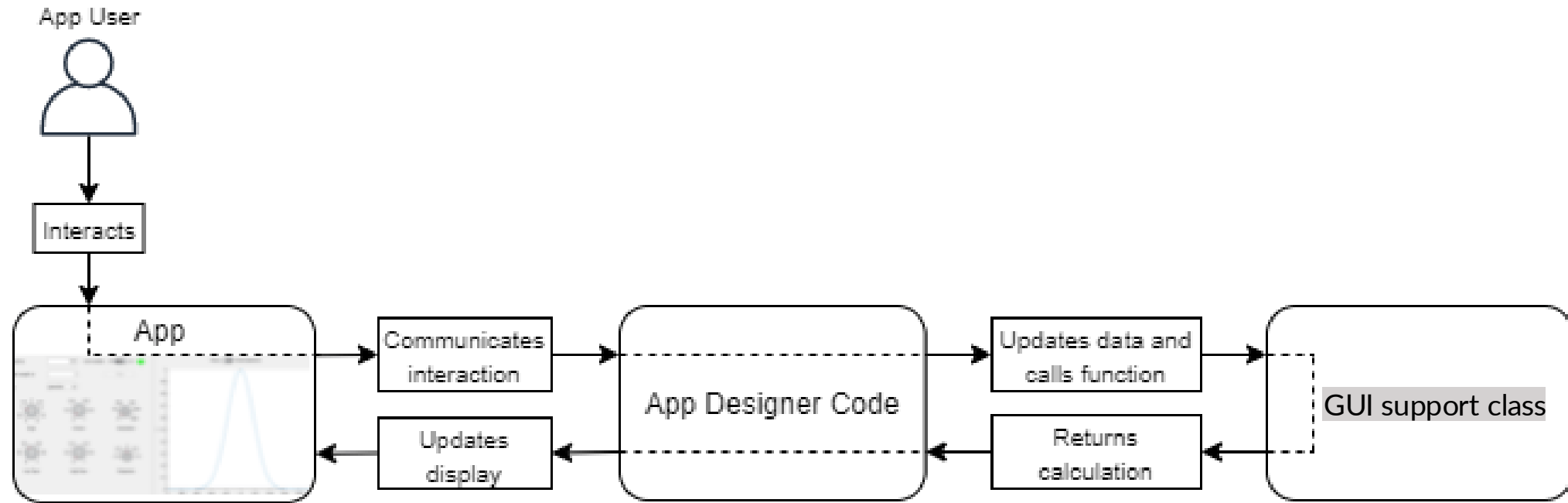
- It is possible to program the entire app in App Designer, but as the app grows, the code becomes hard to maintain

App Designer + Support Class

- Organize the code into separate components with different roles
- More control over structure of the code
- Improved development, testing, and collaboration

Step 2: Create a "GUI support" class definition

Why use a GUI support class?



Step 2: Create a "GUI support" class definition

- Open the template "AppSupportTemplate.m" containing a class definition with the following syntax:

```
classdef (Attributes) ClassName < SuperClassNames
    properties (Attributes) ... end
    methods (Attributes) ... end
    events (Attributes) ... end
end
```

- Classes allow us to create objects for use in object-oriented programming
- This "GUI support" class definition will store and manage the application data
- It will also communicate with the app to make updates when data changes occur

Step 2: Create a "GUI support" class definition

- The class needs to inherit from the Superclass "handle"

```
classdef (Attributes) ClassName < handle
```

- For more information on the handle superclass, refer to: <https://www.mathworks.com/help/matlab/ref/handle-class.html>
- For more general information about MATLAB classes, refer to: <https://www.mathworks.com/help/matlab/object-oriented-programming.html>

Step 2a: Add properties, events, and listeners

Properties

- Properties contain data that is important and relevant to the object
- Can be public or private, constant, dependent on other values

The template has the following properties:

```
properties (Constant)
```

```
    numPlotPts = 50;
```

```
end
```

```
properties
```

```
    pvlist PV
```

```
    pvs
```

```
    guihan
```

```
    plotOptionState
```

→ % Add a property that stores application data here

```
end
```

Step 2a: Add properties, events, and listeners

Events and Listeners

- Events are broadcast when an action or change occurs to the object
- Listeners execute a callback when the event is triggered

Event defined in the template:

```
events
```

```
  PVUpdated
```

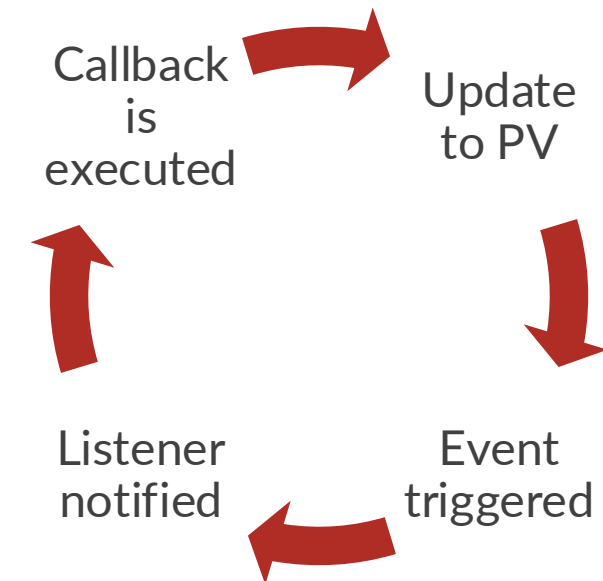
```
end
```

Listener defined in the template:

```
properties (Hidden)
```

```
  listeners
```

```
end
```



Step 2b: Create a constructor method

- The constructor method creates an instance of the class
- It can also initialize property values and call other methods
- For more information about constructor methods: https://www.mathworks.com/help/matlab/matlab_oop/class-creator-classes.html

Step 2b: Create a constructor method

methods

```
function obj = AppSupportTemplate(apphandle)

    % Initialize object and add PVs to be monitored
    context = PV.Initialize(PVtype.EPICS);
    obj.pvlist = [...      % Associates PV with App component
        PV(context, 'name', "GaugePV1", 'pvname', "SIOC:SYS1:ML00:A0951", 'mode', "rw",
            'monitor', true, 'guihan', apphandle.PV1Gauge);
    % Add the second PV here ];

    pset(obj.pvlist, 'debug', 0);
    obj.pvs = struct(obj.pvlist);
    obj.guihan = apphandle;
```

Step 2b: Create a constructor method

% Create arrays to store the PV values to be plotted at each time stamp. Use the property numPlotPts, and store both arrays in the data property you defined above. Add your code here:



```
% Set the initial state of the app to "Waiting for Input"
```

```
obj.data.plotOpt = "Waiting For Input";
```

```
% Start listening for PV updates
```

```
obj.listeners = addlistener(obj, 'PVUpdated', @(~,~) obj.loop);
```

```
run(obj.pvlist, false, 0.1, obj, 'PVUpdated');
```

```
end
```


Step 2c: Create a loop method

- Create a function that the script loops through every time the PVs are updated
- Make sure that the data stays updated
- Ignore the switch in the template for now – we will fill this in later

Step 2c: Create a loop method

```
function loop(obj)
```

```
    PV1_val = obj.pvs.GaugePV1.val{1};
```

```
    PV2_val = obj.pvs.GaugePV2.val{1};
```

```
    % Assign each variable to the last value in its respective PV array:
```



```
    % Use “circshift” to shift the values in the arrays to the left:
```



```
end
```

Step 2d: Create a "stop" method

- The template defines a method that stops the PV objects from updating

```
function clearPV(obj)
    Cleanup(obj.pvlist);
    stop(obj.pvlist);
end
```

Step 3: Connect App file to GUI support file

- In the App Designer file, go to "Code View"
- Create a new property for the app object

```
properties (Access = private)
```

```
    aobj
```

```
end
```

Step 3: Connect App file to GUI support file

- Create a startup function and create an instance of the GUI support object
 - On the left-hand side in the "CODE BROWSER" panel, click on the "+" button to add a startup function callback

```
function startupFcn(app)
    app.aobj = AppSupportTemplate(app);
end
```

Step 3: Connect App file to GUI support file

- Additionally, add a Close Request function

```
function UIFigureCloseRequest(app,event)

    try

        app.aobj.clearPV();

    catch

        delete(app);

    end

    delete(app)

end
```

Step 4: Add callbacks

- On the right-hand side in the "COMPONENT BROWSER" panel, right click on the Close App button and add a callback
- Call the Close Request function

```
function CloseButtonPushed(app, event)
    app.UIFigureCloseRequest()
end
```

Step 4: Add callbacks

- On the right-hand side in the "COMPONENT BROWSER" panel, right click on the Drop-Down and add a callback
- When the user chooses an option from the Drop-Down menu, the callback should send a command to the GUI support file
 - The App should tell the GUI support object to "do something," and the object should store/update the new state of the App
 - If the user chooses "PV1," the GUI support object should update the current state to "PV1" and plot the values for PV1
 - If the user chooses "PV2," the GUI support object should update the current state to "PV2" and plot the values for PV2

Step 4: Add callbacks

- In the GUI support file, add functions that update the state of the object (Plot Option State) if a new Drop-Down option is selected by the user

```
function % Name your function ←  
    cla(obj.guihan.UIAxes);  
    % Add your code here:  
→  
end
```

- Repeat for PV2

Step 4: Add callbacks

- In the App file, add a switch to the Drop-Down callback

```
function PVDropDownValueChanged(app,event)
```

```
    value = app.PVDropDown.Value;
```

```
    switch value
```

```
        case 'PV1'
```

```
        → % Add your function from the last part here
```

```
        case 'PV2'
```

```
        → % Add your function from the last part here
```

```
    end
```

```
end
```

Step 4: Add callbacks

- In the GUI support file, add a switch to the loop method

```
function loop(obj)
```

```
    PV1_val = obj.pvs.GaugePV1.val{1};
```

```
    PV2_val = obj.pvs.GaugePV2.val{1};
```

```
    % Assign each variable to the last value in its respective PV array:
```

```
    % Done in Part 2c
```

```
    switch % Switch variable here ←
```

```
        case % Case where PV1 is chosen ←
```

```
            → % Plot PV1 values here
```

```
                drawnow
```

Step 4: Add callbacks

```
case % Case where PV2 is chosen ←
```

```
→ % Plot PV2 values here
```

```
drawnow
```

```
otherwise
```

```
% Do nothing. This case is only true when the app is first launched, and the  
state is "Waiting for Input"
```

```
end
```

```
% Use "circshift" to shift the values in the arrays to the left:
```

```
% Done in Part 2c
```

```
end
```

Run your app!

Summary

- App Designer is a great tool to make GUIs in Matlab
- Use a support class ("the model") for better organization and improved development, testing, and collaboration
- Use callbacks to make your GUI interactive