Let’s Build a Simple App! 🎉

Goal: Create a “Get Current Time” App ⏰

Hidden Track: Secret Sauce for Building GUIs Like a Pro!

Leverage the tips and tricks from past weeks

Week 9: GUI Development and Symbolic Math 🖥️🔢

Objectives:

- Introduction to GUI development using App Designer 🛠️.
- Utilize symbolic math capabilities 📐.
- Perform differentiation and integration ✏️.

Topics Covered:

- GUI Development with App Designer 🖥️
- Symbolic Math 📊

GUI Development with App Designer 🖥️

What are GUIs?

Graphical User Interfaces (GUIs) provide a user-friendly way to interact with programs.

🧩 **Components Include**:

- Buttons
- Text fields
- Drop-down menus
- Sliders
- Plots

Why Use GUIs?

- Enhance user experience 🎨
- Make programs accessible to non-programmers 👥
- Visualize data dynamically 📈

Introducing App Designer

MATLAB's App Designer is a drag-and-drop environment for building professional apps.

🛠️ **To Open App Designer**:

```
appdesigner
```

- Generates a `.mlapp` file containing your app's layout and code.

💡 Tip: Save your work frequently to prevent data loss!

General Form for GUI Objects

- `app` - The application as a whole (handle)
- `app.element` - Some element of the application (handle)
- `app.element.Value` - The value of some element (text, input, selection, etc)

Analogy

A *handle* is a reference to an object, a property is simply a property like “Value”. Think of it like a “link” to a website, it’s not the website itself you’re sending, just a pointer to the website

Important Function Difference

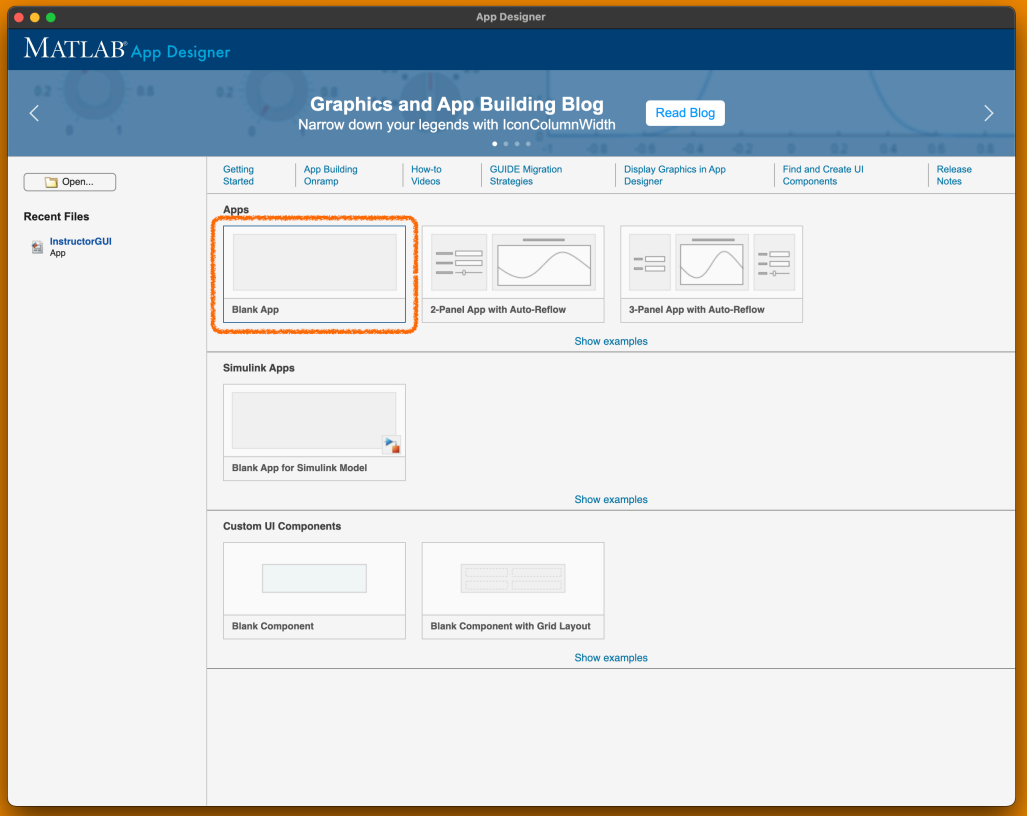
Many functions in an app, require a handle so they know where to apply that function.

Examples:

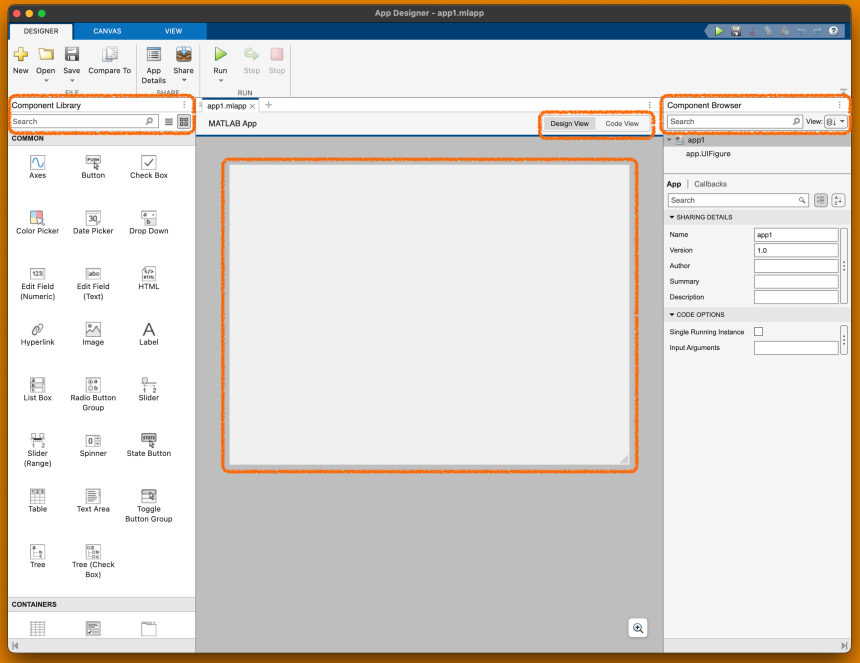
- `plot(app.pltLinear, x, y)`
- `xlabel(app.pltLinear, “text”)`

Step 1: Set Up the App

1. Open App Designer.
2. Select Blank App.

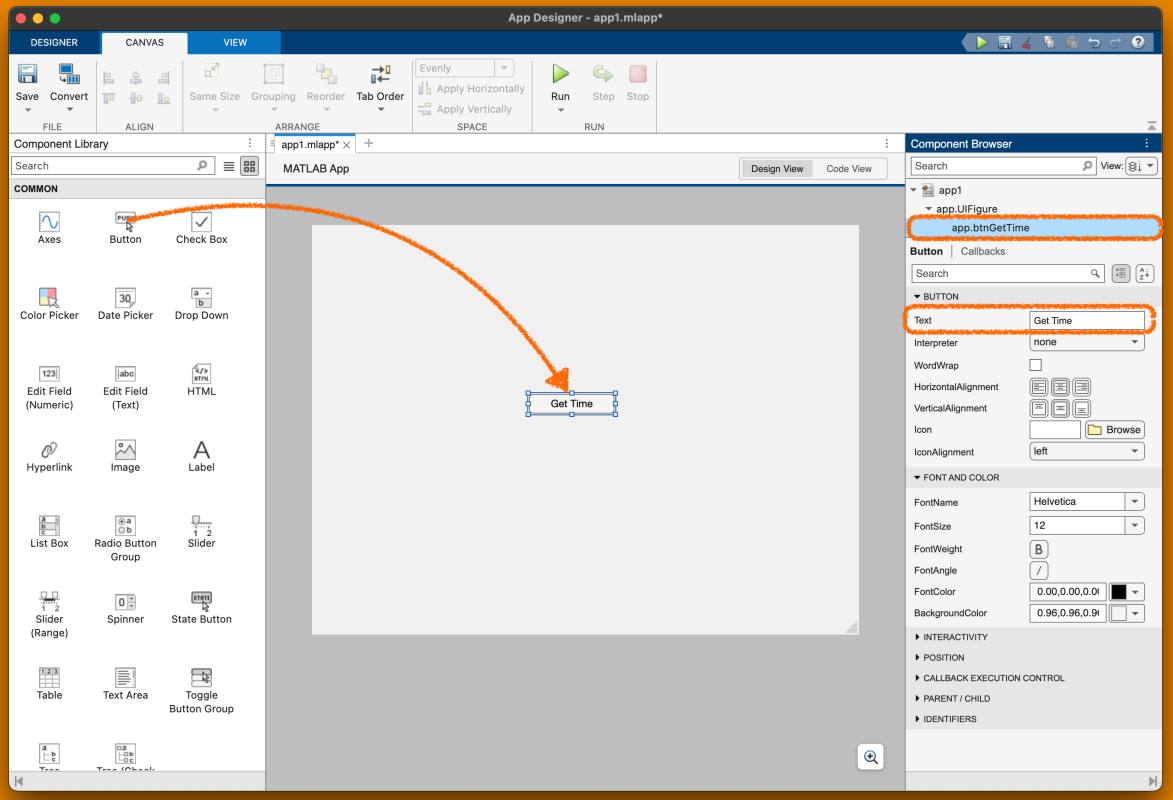


🔎 Have a look around 👀



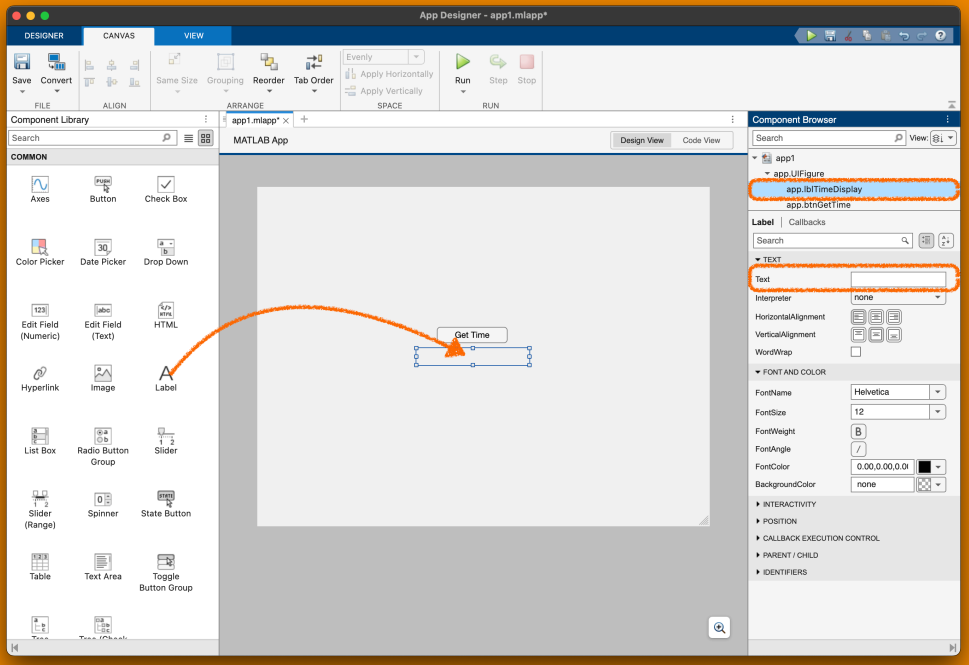
Step 2: Design the Interface

1. Drag a Button onto the canvas.
2. Customize the button:
   - Text: Get Time
   - Component Name: `btnGetTime`



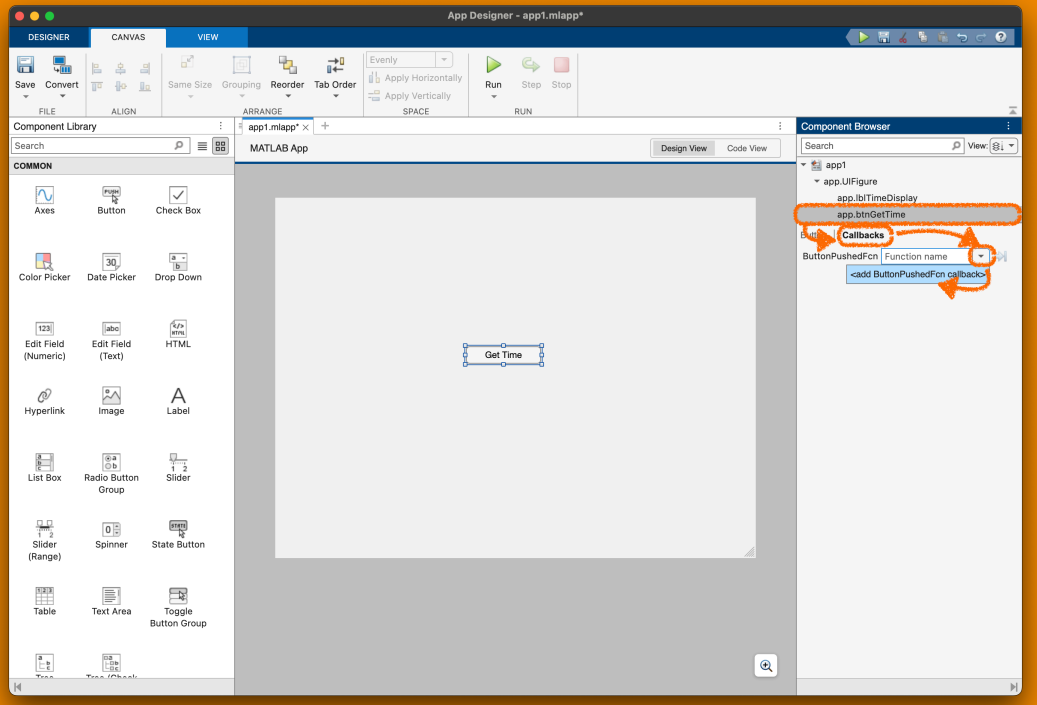
Step 3: Add a Display Field

1. Drag a Label onto the canvas below the button.
2. Clear the default text (we'll update it programmatically).
3. Component Name: `lblTimeDisplay`



Step 4: Add Callback to Button:

1. Select the button.
2. In the Component Browser, expand Callbacks.
3. Click Add Callback > ButtonPushedFcn.



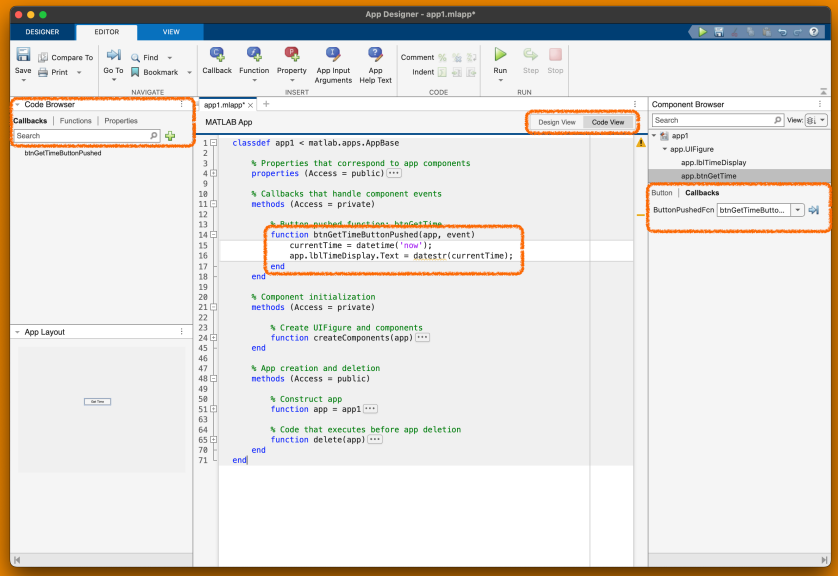
A callback is a function executed in response to a function or in this case; user action.

Step 5: Implement the Logic

```
% Button pushed function: btnGetTime
function btnGetTimePushed(app, event)
    currentTime = datetime('now');
    app.lblTimeDisplay.Text = datestr(currentTime);
end
```

Explanation:

- Retrieves the current time.
- Updates the label to display the time.



Step 6: Run and Test Your App

- Click Run ▶️.
- Test the button to see the current time display.

🎉 Congratulations! You've built your first app!

Deep Dive into Callbacks 🔍

What Happens When You Click the Button?

- The `btnGetTimePushed` function is called.
- This function defines the app's behavior upon interaction.

Anatomy of a Callback Function

```
function btnGetTimePushed(app, event)
    % Your code here
end
```

- `app`: Reference to your app's components.
- `event`: Data related to the event (e.g., mouse click).

Best Practices for Callbacks

- Keep them concise.
- Call other functions for complex tasks.
- Use descriptive names.

Creating Helper Functions

- Use helper functions to organize code.

Example:

```
methods (Access = private)
    function updateTimeDisplay(app)
        currentTime = datetime('now');
        app.lblTimeDisplay.Text = datestr(currentTime);
    end
end
```

Call this function within your callback:

```
function btnGetTimePushed(app, event)
    app.updateTimeDisplay();
end
```

Notes:

- Include `app`
- Can return data like a regular function

UserData and Properties

UserData

- Can be used to store information similar to a struct
- Is tied to the component
- Useful for storing information you want to retrieve specific to that component (plot x and y values)
- `app.<componentName>.UserData.<yourVariableName> = 12345`

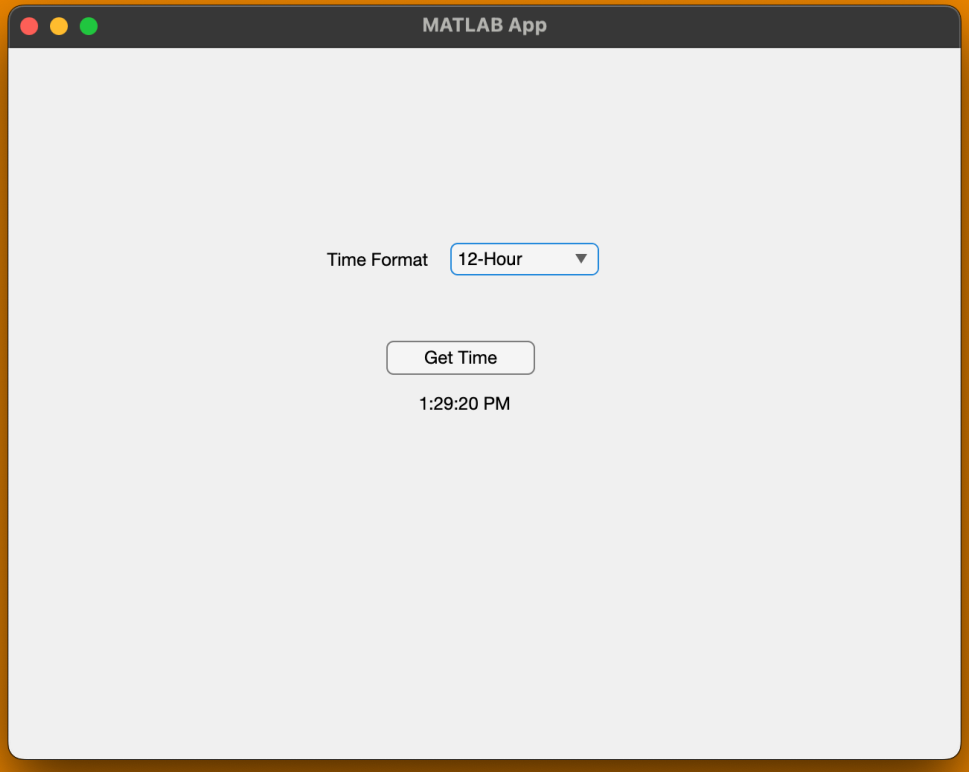
Properties

- Variables globally available anywhere in the GUI code.
- Should be treated like any other global variable

Adding More Features 🌟

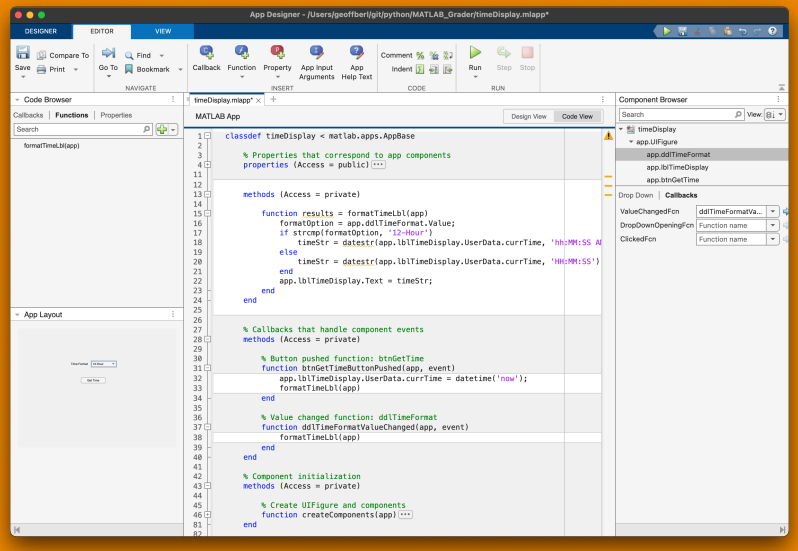
Feature 1: Time Format Options

- Add a Drop-Down Menu to select time formats.
  - Options:
    - 12-Hour
    - 24-Hour



Update the Callback

```
function btnGetTimePushed(app, event)
    currentTime = datetime('now');
    formatOption = app.ddlTimeFormat.Value;
    if strcmp(formatOption, '12-Hour')
        timeStr = datestr(currentTime, 'hh:MM:SS AM');
    else
        timeStr = datestr(currentTime, 'HH:MM:SS');
    end
    app.lblTimeDisplay.Text = timeStr;
end
```

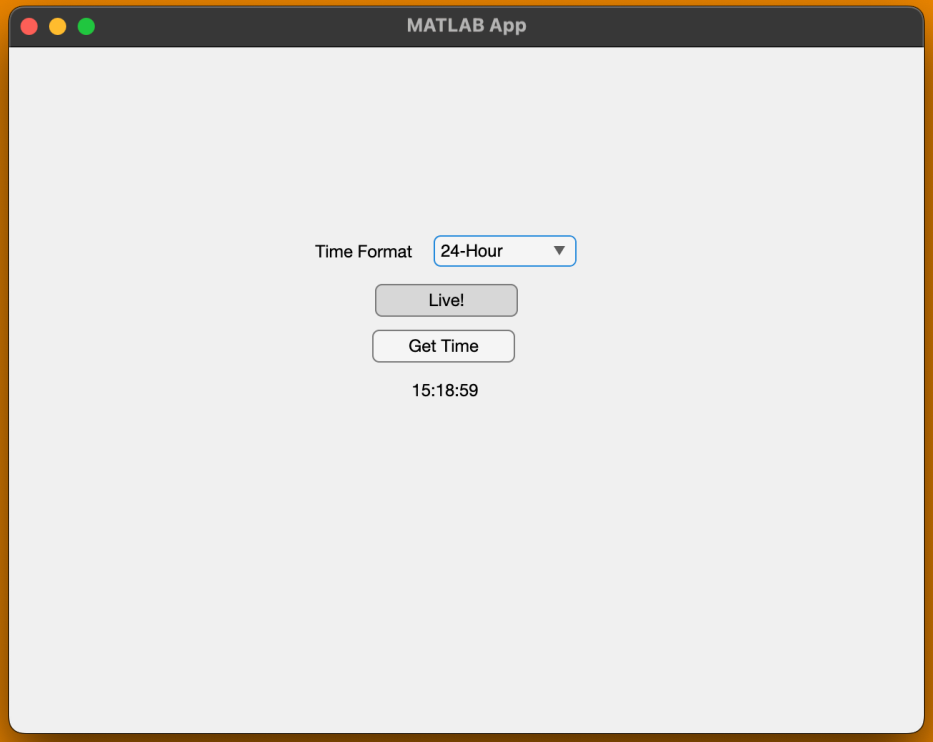


💡 Tips:

- Notice `Value` and `Text`
- Results from `Value` and `Text` are sometimes strings even though the value is numeric

Feature 2: Live Clock

- Add a Switch to toggle a live clock display.
- Use a Timer Object to update the time every second.



Implementing the Timer

```
properties (Access = private)
    Timer % Timer object for live clock
end

% Value changed function: LiveClockSwitch
function LiveClockSwitchValueChanged(app, event)
    if app.LiveClockSwitch.Value == "On"
        app.Timer = timer('ExecutionMode', 'fixedRate', 'Period', 1, ...
            'TimerFcn', @(~,~) app.updateTimeDisplay());
        start(app.Timer);
    else
        stop(app.Timer);
        delete(app.Timer);
    end
end
```

💡Tips:

- Use the debugger to explore and learn
- Clean up the timer (or other resources) when the app closes.

Final Code

```
classdef timeDisplay < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
    ...

    properties (Access = private)
        Timer % Timer object for live clock
    end

    methods (Access = private)

        function formatTimeLbl(app)
            formatOption = app.ddlTimeFormat.Value;
            if ~isfield(app.lblTimeDisplay.UserData, 'currTime')
                btnGetTimeButtonPushed(app, []);
            end
            if strcmp(formatOption, '12-Hour')
                timeStr = datestr(app.lblTimeDisplay.UserData.currTime, 'hh:MM:SS AM');
            elseif strcmp(formatOption, '24-Hour')
                timeStr = datestr(app.lblTimeDisplay.UserData.currTime, 'HH:MM:SS');
            else
                % msgbox("Unhandled selection, please update code")
                fprintf("Unhandled time format (%s), please update code\n", formatOption)
                return
            end
            app.lblTimeDisplay.Text = timeStr;
        end

        function updateTimeDisplay(app)
            btnGetTimeButtonPushed(app, [])
        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: btnGetTime
        function btnGetTimeButtonPushed(app, event)
            app.lblTimeDisplay.UserData.currTime = datetime('now');
            app.formatTimeLbl()
        end

        % Value changed function: ddlTimeFormat
        function ddlTimeFormatValueChanged(app, event)
            app.formatTimeLbl()
        end

        % Value changed function: btnLiveClock
        function btnLiveClockValueChanged(app, event)
            state = app.btnLiveClock.Value;
            if state
                app.Timer = timer('ExecutionMode', 'fixedRate', 'Period', 1, ...
                    'TimerFcn', @(~,~) app.updateTimeDisplay());
                start(app.Timer);
            else
                stop(app.Timer);
                delete(app.Timer);
            end
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
                if ~isempty(app.Timer) && isvalid(app.Timer)
                    stop(app.Timer);
                    delete(app.Timer);
                end
            delete(app);
        end
    end

    % Component initialization
    methods (Access = private)
```

Packaging and Sharing Your App 📦

Exporting the App

- Go to Share > Create Standalone Application.
- Follow the prompts to package your app.

Sharing Options

- MATLAB App Installer (`.mlappinstall`): Share with other MATLAB users.
- Standalone Desktop App: Runs without MATLAB (requires MATLAB Compiler).

Tips for Distribution

- Test your app thoroughly before sharing.
- Include user instructions or a help dialog within your app.

Introduction to Symbolic Math 📐

Why Use Symbolic Math?

- Perform exact computations.
- Solve algebraic equations analytically.
- Simplify expressions.

Getting Started

```
syms x y z
```

Declares symbolic variables.

Differentiation and Integration

Differentiate:

```
f = sin(x) * exp(y);
diff_f = diff(f, x);
```

Integrate:

```
int_f = int(f, x);
```

Visualize Results:

```
fplot(diff_f, [0, 2*pi])
```

Solving Equations

```
eqn = x^2 + 5*x + 6 == 0;
sol = solve(eqn, x);
disp(sol);
```

Output:

```
sol =
  -2
  -3
```

Incorporating Symbolic Math into GUIs 

Example: Symbolic Differentiation App

App Features

- Input an expression to differentiate.
- Select the variable to differentiate with respect to.
- Display the symbolic derivative.

Designing the Interface

- Components:
  - Text Area for input expression.
  - Drop-Down for variable selection.
  - Button to execute differentiation.
  - Text Area to display the result.

Writing the Callback

```
function btnDerivePushed(app, event)
    expr = app.InputExpressionTextArea.Value;
    var = sym(app.VariableDropDown.Value);
    try
        derivative = diff(sym(expr), var);
        app.ResultTextArea.Value = char(derivative);
    catch
        app.ResultTextArea.Value = 'Invalid expression. Please try again.';
    end
end
```

Improving the User Experience (UX)

- Add try-catch blocks to handle invalid inputs.
- Real-Time Updates: Use listeners to update the result as the user types.
- Math Display: Use latex function and display expressions in formatted math.

Displaying Formatted Math

- Utilize the HTML Component to render LaTeX.

Example:

```
latexStr = ['$' latex(derivative) '$'];
app.ResultHTML.Html = ['' latexStr ''];
```

$$e^{y} \cos(x)$$

Key Takeaways 🎓

- App Designer simplifies GUI creation in MATLAB.
- Callbacks are essential for interactive apps.
- Symbolic Math extends MATLAB's capabilities for analytical computations.
- Combining GUIs with symbolic math creates powerful educational tools.

Next Steps 🚀

- Explore More Components: Try sliders, switches, and plots.
- Advanced Features: Look into app packaging and deployment options.
- Join the Community: Share your apps and get feedback.

Questions? 🤔

Feel free to ask any questions or share your thoughts!

Additional Resources 📚

- MATLAB Documentation: App Designer
- Tutorial Videos: MATLAB App Designer Tutorials
- Community Forums: MATLAB Central

✅ Write Logic First, GUI Later:

- Build and test your core logic in a separate script or function.
- Make sure it works without the GUI before you bring it in.

🔧 Use Stubs for GUI Values

- Stub GUI input/output: simulate dropdown values, textbox content, etc.
- This keeps your logic testable without running the full GUI.

🐞 Debug Like a Pro

- Enable Pause on Error/Warning in the MATLAB debugger.
- Use breakpoints strategically to step through critical logic.

🧪 Think Modular

- Keep business logic in separate functions — treat your GUI as a shell, not a brain.

💬 Tip: `disp()` and `fprintf()` Still Work

- Debug print statements are still your friend — even in GUI callbacks!

Example Stub

Assignment 1

```
% FunctionPlotter_Stub.m
% Simulate GUI inputs
userInputExpr = 'x^2 + 3*x + 2';    % ← stubbed from input field
rangeStart = -10;                   % ← stubbed from 'start range' input
rangeEnd = 10;                      % ← stubbed from 'end range' input

% Core logic
try
  syms x
  expr = str2sym(userInputExpr); % Convert input to symbolic
  fplot(expr, [rangeStart, rangeEnd]) % Plot expression
  grid on
  title(['y = ', char(expr)])
catch err
  disp('Invalid expression!');
  disp(err.message);
end
```

<