App Designer for MATLAB®

User's Guide

R2014**b**— Tech Preview

MATLAB®

MathWorks®

App Designer for MATLAB® User's Guide

Revision History

October 2014	Online only	New for MATLAB Version 8.4 (Release 2014b — Tech Preview)

Contents

3 App Programming

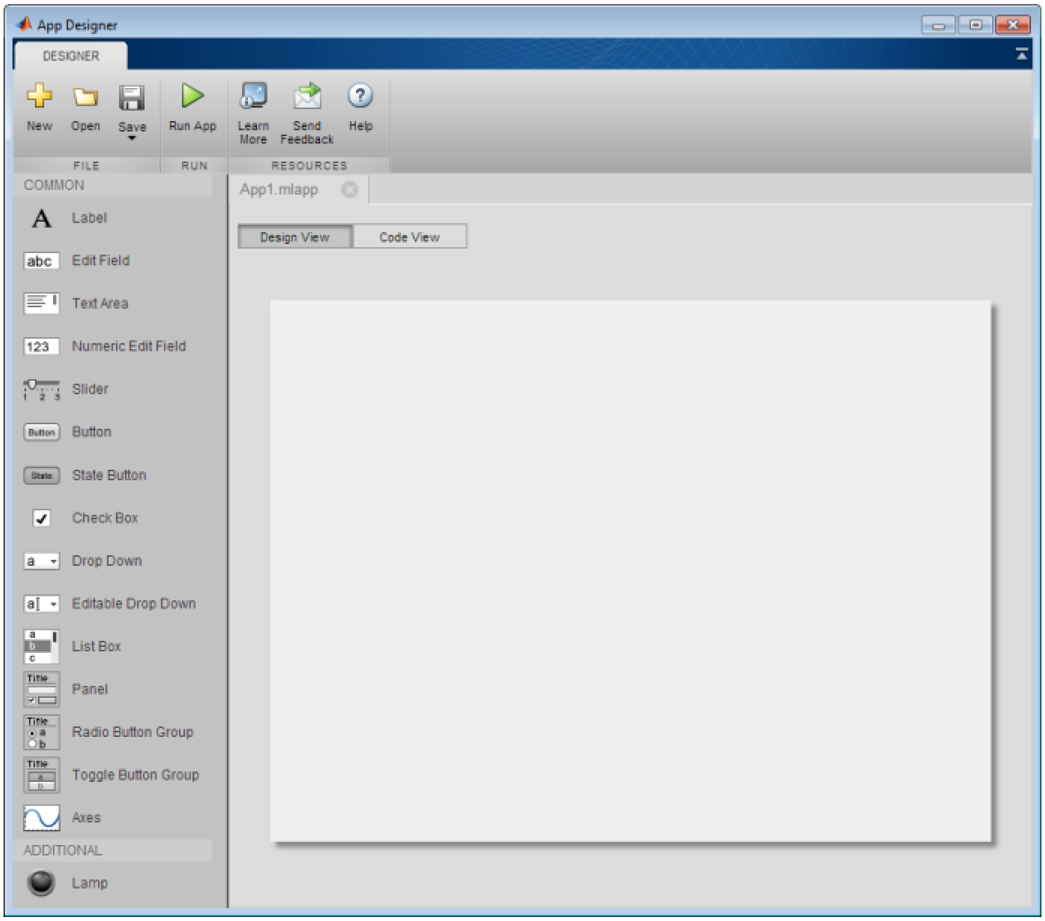
4 Keyboard Shortcuts

App Designer Basics

Open App Designer

In the MATLAB® Command Window, type:

```
appdesigner
```



To open a previously created app with App Designer, type the following, where *filename* the name of an app file (.mlapp) on the MATLAB search path:

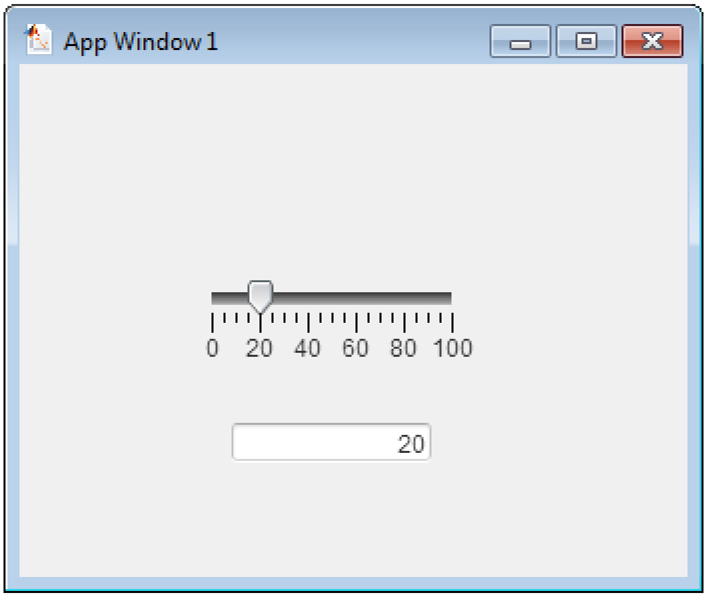
```
appdesigner filename
```

App Creation and Modification

In this section...
"Create and Run a Simple App" on page 1-3
"Run an Existing App" on page 1-4
"Edit an Existing App" on page 1-5

Create and Run a Simple App

This example shows how to create and run a simple app. The app displays the value of a slider after the user moves the slider bar to a new location.



1. Open App Designer.

   In the MATLAB Command Window, type `appdesigner`.
2. From the App Designer component palette on the left, drag a Slider and a Numeric Edit Field into the central design area.

3. In the central design area, right-click the slider and select **Callbacks** > **Edit ValueChangedFcn**.
4. In the Edit ValueChangedFcn Callback dialog box, accept the default name by clicking **OK**.

   Although not done here (to keep the example simple), changing the function name makes it easier to associate the callback code with a component, particularly when there are multiple instances of the same component in an app.
5. In the code file that opens, add the following code under `function SliderValueChanged(app)`. The code directs MATLAB to update the numeric edit field to reflect the slider bar location.

   ```
   app.NumericEditField.Value = app.Slider.Value;
   ```

   Your code can access all components and their values directly, similar to the way that you access fields of a structure array. To access or set a property value, specify the app that you are designing (which is always `app`), the component, and the component property name.
6. On the **Editor** tab, click **Run App**.

   Save the app when prompted.

Test the app by changing the slider value and releasing the mouse button. The numeric edit field reflects the new value.

“App Displaying Slider Value as it Changes” on page 3-27, updates the edit field continuously as the slider is moved.

Run an Existing App

To run an app you created previously:

1. Open App Designer:

   ```
   appdesigner
   ```
2. On the **Designer** tab, click **Open**.
3. In the Open dialog box, navigate to the folder containing your MLAPP file and specify the name in the **File name** field.
4. Click **Open**.

App Designer displays the app in design view.

5. On the **Designer** tab, click **Run**.

Alternatively, you can type the file name (*excluding* the `.mlapp` file extension) in the MATLAB Command Window. The file must be in a folder on your MATLAB path.

Edit an Existing App

To open App Designer and edit an existing app:

1. Open App Designer

   ```
   appdesigner
   ```
2. On the **Designer** tab, click **Open**.
3. In the Open dialog box, navigate to the folder containing your MLAPP file and specify the name in the **File name** field.
4. Click **Open**.

   App Designer displays the app in design view.
5. Click **Code View** to view and edit the app code.

Alternatively, in MATLAB Command Window, type this command, where *filespec* is the file name, full file path, or partial file path of the app file (`.mlapp`):

```
appdesigner filespec
```

If you type just the file name, the file must be in a folder on your MATLAB path.

Uninstall App Designer Tech Preview

To uninstall the App Designer tech preview software and documentation, type this command in the MATLAB Command Window:

```
uninstallappdesigner
```

App Layout

Add and Delete Components

When you first open App Designer, the design area, which represents your app, is empty. To populate it with components, drag components from the component palette (on the left) into the design area (in the middle).

To add or remove components, you can then:

- Cut, copy, paste, or delete a single component in the design area.

  Right-click the component and select an option from the context menu.
- Cut, copy, paste, or delete multiple components in the design area.

  Multiselect the components, right-click, and then select an option from the context menu.
- Duplicate a component.

  Right-click the component and drag the duplicate to a new location within the current app.

Note: When you duplicate or copy and paste components, the new components have the same property values as the original components, with the exception of callbacks, **Name in Code** and **Position** properties. Those properties are set to their default values.

You can copy or cut components from one app and paste them in another.

Component Selection

Component selection is the first step in performing many operations.

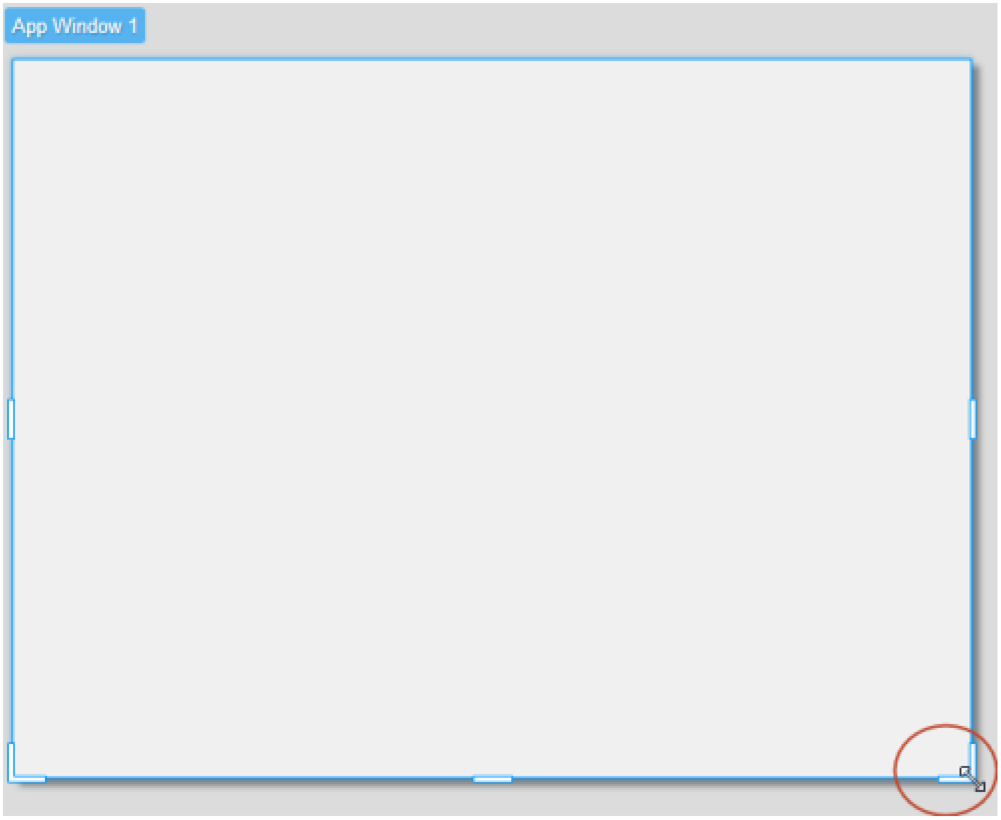
Selection	How To
Single component	Click the component.
Multiple components	Hold down the **Shift** key as you click each component that you want to include in the selection. Or Click an empty region in the design area, and then drag a selection border around all the components that you want to select. App Designer selects only components completely contained within the selection.
All selectable components, excluding those that are children of another component.	Press **Ctrl+A**.
Unselect a previously selected component or reselect an unselected component (toggle selection).	Press **Shift+Click**.

To move the selection to the next component, container, or app window in the design area's tab key navigation sequence, press the **Tab** key. To move the selection to the previous component, container, or app window in the tab key navigation sequences, press **Shift+Tab**.

Resize App Window

To resize the app window, click its border in the design area, and then drag an expander.

To maintain the current aspect ratio, place the cursor over the expander. Hold down the **Shift** key as you drag the expander.



To cancel an in-process resize operation, press the **Escape** key.

Move Components

You can move components manually (freehand) or by set increments, as follows:

- Freehand — Drag the component or components.
- 1-pixel increments — Select the component or components. Press a keyboard **Arrow** key in the direction that you want to move the component.
- 10-pixel increments — Select the component or components. Press **Shift+Arrow** in the direction that you want to move the component.

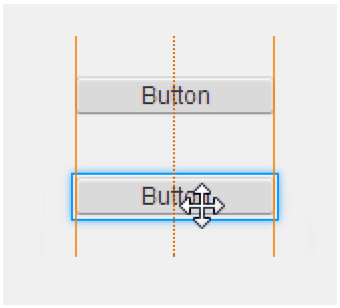
To cancel an in-process freehand move operation, press the **Escape** key.

Align Components

You can direct App Designer to align components automatically, or you can align them manually. For instance, you can use the manual method for an initial layout, and then polish the layout using automatic alignment.

Manual Component Alignment

As you drag components in the design area, you see alignment hints that assist you in arranging the components.



- Dashed orange lines indicate that components are center-aligned.
- Solid orange lines indicate that components are aligned along their edges (top, bottom, or sides).

Alignment hints are not displayed if you select a component and then press arrow keys to move the component.

Automatic Component Alignment

To use the automatic alignment feature:

1. In the design area, select two or more components.
2. On the **Layout** tab, in the **Alignment** section, click a control to indicate how you want those components to align.

Space Components

You can direct App Designer to space components automatically, or you can manually specify spacing, as follows:

1. In the design area, select the components that you want to space.

   You must select two or more components to space manually and three or more components to space automatically.
2. On the **Layout** tab, in the **Spacing** section, do one of the following:

   - For manual spacing, select **Use .... pixels** and type a value in the **pixels** field.
   - For autospacing, select **Autospacing**.

     App Designer autospaces by:

     - Evenly distributing components between the top and bottom components when you apply vertical spacing.
     - Evenly distributing components between the leftmost and rightmost components when you apply horizontal spacing.

     Sometimes, the number of pixels between the two components prevents App Designer from making the space between each component identical. However, the spacing difference is never more than 1 pixel.
3. Select how you want spacing applied by selecting **Apply horizontally** or **Apply vertically**.

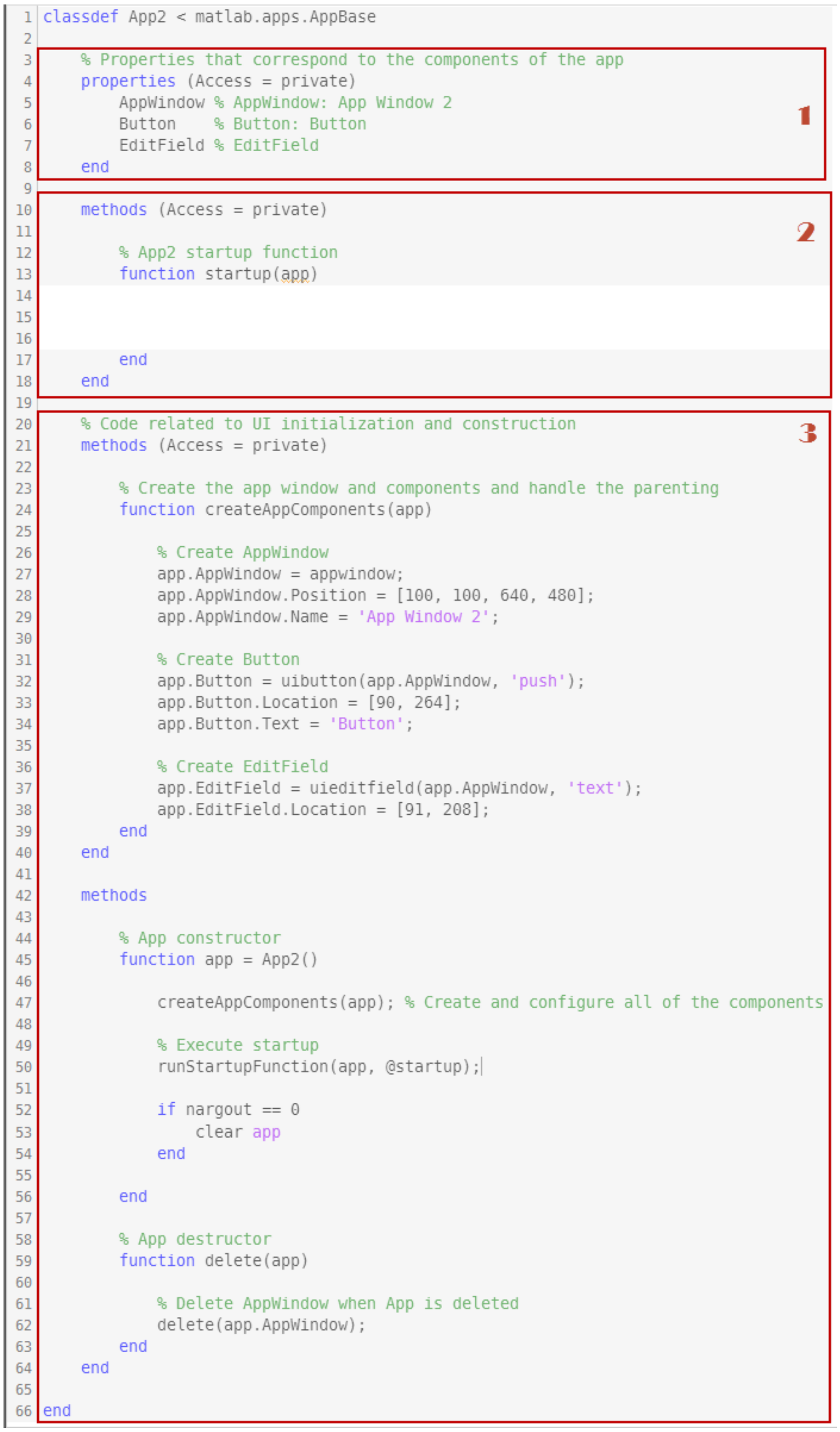
App Programming

App Designer Code Generation

As you create an app, App Designer generates code to reflect your component layout and customizations. In addition, it generates the framework for the callback functions, which control app behavior. When you save your app, App Designer saves the code in an MLAPP file. When MATLAB runs your app, it executes this file.

App Designer autogenerates and manages the code for creating the app, its components, and the framework for functions. To prevent you from accidentally overwriting the code that App Designer manages, that code is read only (indicated with a light gray background).

The code that App Designer autogenerates is modularized. For example, when you add a button and an edit field to the design area the generated code is as shown in the image that follows.



```
classdef App2 < matlab.apps.AppBase

    % Properties that correspond to the components of the app
    properties (Access = private)
        AppWindow % AppWindow: App Window 2
        Button    % Button: Button
        EditField % EditField
    end

    methods (Access = private)

        % App2 startup function
        function startup(app)



        end
    end

    % Code related to UI initialization and construction
    methods (Access = private)

        % Create the app window and components and handle the parenting
        function createAppComponents(app)

            % Create AppWindow
            app.AppWindow = appwindow;
            app.AppWindow.Position = [100, 100, 640, 480];
            app.AppWindow.Name = 'App Window 2';

            % Create Button
            app.Button = uibutton(app.AppWindow, 'push');
            app.Button.Location = [90, 264];
            app.Button.Text = 'Button';

            % Create EditField
            app.EditField = uieditfield(app.AppWindow, 'text');
            app.EditField.Location = [91, 208];
        end
    end

    methods

        % App constructor
        function app = App2()

            createAppComponents(app); % Create and configure all of the components

            % Execute startup
            runStartupFunction(app, @startup);

            if nargout == 0
                clear app
            end

        end

        % App destructor
        function delete(app)

            % Delete AppWindow when App is deleted
            delete(app.AppWindow);
        end
    end

end
```

Notice that:

- The first section of code defines the app properties.

  Initially, this section contains the properties that correspond to the app and the app components. If you add properties to share data, App Designer adds your properties to this region. For information on adding properties, see “Share Data Across Callbacks” on page 3-11
- The second section of code contains the app functions.

  Initially, this section contains the framework for the startup function. If you add callbacks or other functions to the app, App Designer adds them to this region. For information on adding functions, see “Callbacks — Programmed Response to User Actions” on page 3-6 and “App Functions” on page 3-13.
- The third section of code initializes and creates the app and its components.

  If you select a component in design view and use the **Properties** tab to change property values for that component, App Designer reflects those changes in this section of the code.

If you rename a component,property, or function (including a callback function), App Designer updates the name in the read-only and read-write areas of the code.

Component Name in App Code

App Designer references components in the code using the component name. If there is more than one of a given component, App Designer appends the component name with a numeric value. For example, if you include three push buttons in your app, App Designer names them `Button`, `Button2`, and `Button3`.

The best practice is to rename each component with a name that helps you to identify it when you read the code. For instance, if a push button causes the app to display results, you might rename it `ResultsButton`. If you create an app and accept the default names, but later decide to change them, you can do so, as described in “Rename Components, Properties, Callbacks, and Functions” on page 3-16.

Callbacks — Programmed Response to User Actions

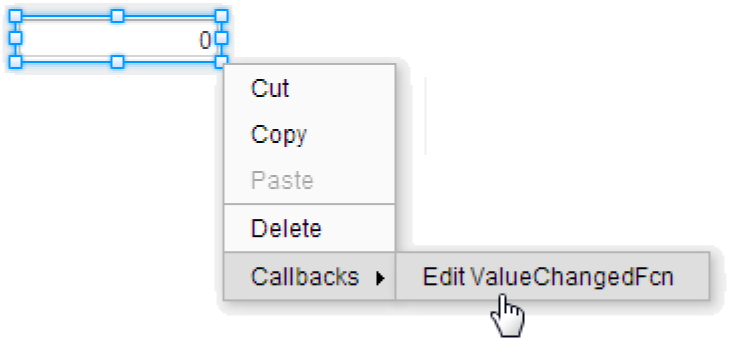
A callback is a function that executes in response to some predefined user action, such as pressing a button or entering a value in a numeric edit field.

Most app components support callbacks. Labels, axes, radio buttons, toggle buttons, lamps, and gauges do not. (However, button groups that contain radio or toggle buttons do have callback functions.)

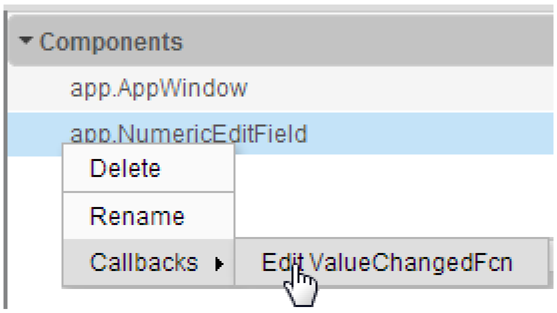
Callback Creation

You can create a component callback from design view or code view:

- To create a callback from design view, right-click the component and select **Callbacks** > **Edit** callback-name.



- To create a callback from code view, in the code browser, right-click the component and select **Callbacks** > **Edit** callback-name.



Then, follow these steps:

1 In the Add Callback Function dialog box, enter a name for your callback and click **OK**.

   App Designer displays code view.

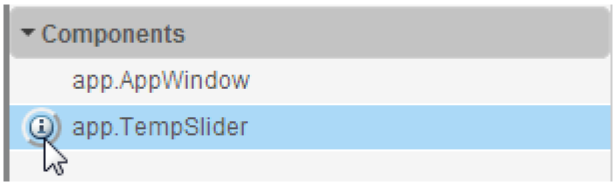
   A best practice is to name the callback so that you can easily identify the control with which it is associated in the app code and what the callback does. For instance, if you intend for a button push to result in data being plotted, you might name the button callback `PlotButtonPushed`.

2 At the location in the code where App Designer places the cursor, type the code that you want MATLAB to run when the user manipulates the control.

   Refer to components and their properties using `app.` as a prefix for each component. This makes it possible to access properties directly, similar to the way you access fields of a structure. For example, to set the Enabled property of the slider named `TempSlider` to `'off'`, use the following code:

```
app.TempSlider.Enabled = 'off';
```

Tip To view component property names and a brief description of each, click the info icon next to a component in the code browser.



Complete property descriptions are available in Chapter 5.

3 Run your app.

Test to ensure that manipulating the component has the effect you want. If your app returns errors, debug your code as described in “Debug App Code” on page 3-10.

Note: Multiple components cannot share a callback. If you want to share code among callbacks, consider writing a function and calling it from multiple callbacks, as described in “App Functions” on page 3-13.

For detailed callback examples, see “App Examples” on page 3-19.

App Initialization

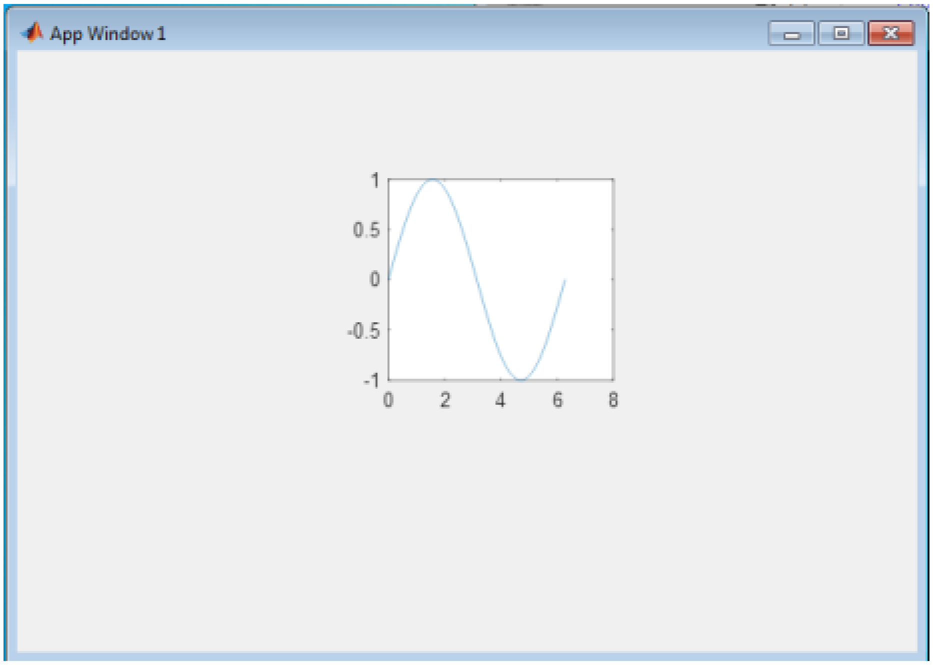
Use the app startup function to perform tasks that you want MATLAB to perform when the app first opens. MATLAB runs the startup function after creating the components.

For instance, use the startup function if you want an axes to appear with a default plot when the app opens or if you want to initialize app properties.

To access the startup function in your app, from code view, click **View Startup Function** on the **Editor** tab.

Display Plot in Axes When the App Opens

This example shows how to create an app that opens with a plot in an axes.



In App Designer:

1. Drag an axes into the design area.
2. Click **Code View**.
3. On the **Editor** tab, click **View Startup Function**.

   App Designer places your cursor under `function startup(app)`.
4. Add the following code:

   ```
   x = linspace(0,2*pi,100);
   y = sin(x);
   plot(app.Axes,x,y);
   ```
5. Save and run the app.

   When the app opens, the app displays the axes with the specified plot.

Debug App Code

Many of the debugging features available in the MATLAB Editor are also provided in the App Designer code view. In particular, you can:

- Set and clear standard breakpoints.
- Step through a file, pausing at points where you want to examine values.

  After you set one or more breakpoints and click **Run App**, the App Designer **Editor** tab displays **Continue**, **Step**, and **Quit Debugging** buttons.
- View variable values in the MATLAB Workspace browser.
- Clear a breakpoint by clicking it.

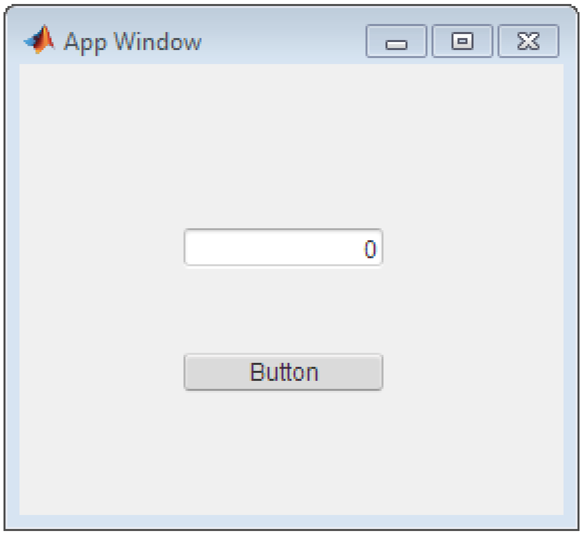
For details on these debugging features, see the “Debugging Process and Features” topic in the MATLAB documentation.

Share Data Across Callbacks

Create properties to store data that you want to share across callbacks. For instance, you can create a property to hold the result of a calculation. App Designer prefixes properties that you create with `app.`, which make it possible to access properties directly, similar to the way you access fields of a structure array.

Store and Display Count of Button Presses

This example shows how to create an app that maintains a count of how many times a button is pressed during an app session.



1 Drag a **Button** and a **Numeric Edit Field** into the design area.
2 Click **Code View**.
3 In the code browser, under **App Properties**, click **Add property**.
4 In the edit field that opens, type `Counter`.

   App Designer adds the `app.` prefix to `Counter` and lists your property in the code browser.
5 Initialize the counter property.

   a On the tool strip **Editor** tab, click **View Startup Function**

   b. In the code, on the line following `function startup(app)` type:

      ```
      app.Counter = 0;
      ```

6. In the code browser, right-click **app.Button** and select **Callbacks** > **Edit ButtonPushedFcn**.
7. In the Add Callback Function dialog box, click **OK** . (Accept the default callback function name.)
8. Add code to increment the counter and display the counter value in the numeric edit field each time the user clicks the push button.

   Immediately after the `function ButtonButtonPushed(app)` line, add:

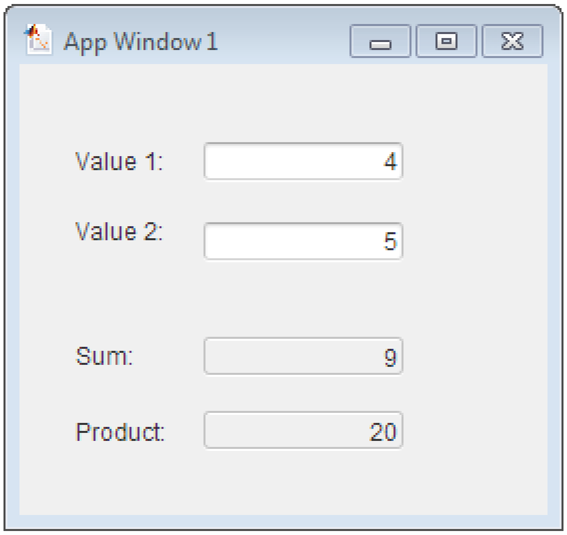
   ```
   app.Counter = app.Counter + 1;
   app.NumericEditField.Value = app.Counter;
   ```
9. Save, run, and test the app.

App Functions

In addition to coding callback functions for your app, you can create app functions that are independent of any particular component. For example, you can create an app function to perform a calculation. You can then call that function from multiple callback functions, eliminating the need to repeat that code in each callback that uses the calculation.

App to Perform a Calculation

This example shows how to create an app that performs a calculation. It uses a function, `mycalc`, to calculate the sum and product of two values specified by the user. When the user changes either value, the app recalculates the sum and product by calling `mycalc`.



In App Designer:

1 Add four numeric edit fields and labels to the design area, as shown in the preceding image and described in this table.

Component	Associated Label Text	Component Name in Code	Nondefault Property Values
Numeric Edit Field	Value 1:	`Value1`	None

Component	Associated Label Text	Component Name in Code	Nondefault Property Values
Numeric Edit Field	Value 2:	`Value2`	None
Numeric Edit Field	Sum:	`Sum`	**Minimum:** `-Inf` **Maximum:** `Inf` **Editable:** Clear this check box.
Numeric Edit Field	Product:	`Product`	**Minimum:** `-Inf` **Maximum:** `Inf` **Editable:** Clear this check box.

2 Click **Code View**.

3 Create the function to calculate the sum and the product of the values that the user enters for value 1 and value 2:

   a In the code browser, click **Add function**.

   b In the add function field, type the function name, `mycalc`.

   c Add code to get the user-entered values, perform calculations, and assign a vector to the function output argument, `result`.

   Immediately after the `function result = mycalc(app)` line, add:

```
a = app.Value1.Value;
b = app.Value2.Value;
sum = a+b;
prod = a*b;
result = [sum,prod];
```

4 Code the callback for the Value 1 numeric edit field to call the `mycalc` function whenever the user changes the field value:

   a In the code browser, right-click **app.Value1** and select **Callbacks** > **Edit ValueChangedFcn**.

   b In the Add Callback Function dialog box, change the name to `Value1Changed`, and then click **OK**.

c. In the code, beginning with the line after `function Value1Changed(app)`, add the following code:

```
calc = mycalc(app);
app.Sum.Value = calc(1);
app.Product.Value = calc(2);
```

5. Code the callback for the Value 2 numeric edit field to call the `mycalc` function whenever the user changes the field value:

   a. In the code browser, right-click **app.Value2** and select **Callbacks** > **Edit ValueChangedFcn**.
   b. In the Add Callback Function dialog box, change the name to `Value2Changed`, and then click **OK**.
   c. In the code, beginning with the line after `function Value2Changed(app)`, add the following code:

   ```
   calc = mycalc(app);
   app.Sum.Value = calc(1);
   app.Product.Value = calc(2);
   ```

6. On the **Editor** tab, click **Run App**.

   Save the app when prompted.

Test the app by entering values for Value 1 and Value 2.

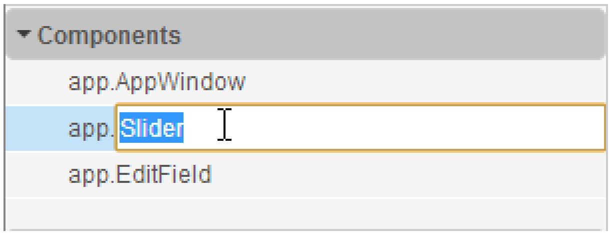
Rename Components, Properties, Callbacks, and Functions

You can rename components, properties, and functions (including callbacks) in your app. App Designer propagates the name change throughout the app code. You can rename any of these app elements from code view. From design view, you can rename app components only.

Rename Elements from Code View

To rename a component, property, or function from code view:

1. In the code browser, double-click the code name.



2. In the edit field that opens, type a new name.

   App Designer updates the code to reflect the component's new name when you press **Enter** or click away from the edit field.

   You cannot change the `app.` prefix. All app component names must have the `app.` prefix.

Tip If you are unsure which name corresponds to a particular component, use the thumbnail view of the app that appears below the code browser. Click the component that you want to identify. App Designer highlights the name of that component in the code browser.

Rename Components from Design View

1. In the design area, select the component for which you want to change the code name.
2. Click the **Properties** tab.

The **Name in Code** field displays the name by which that component is referenced in the code.

3. In the **Name in Code** field, type a new name.

   Press **Enter** or click away from the field. App Designer updates the code to reflect the component's new name.

Delete Components, Properties, Callbacks, and Functions

You can delete components, properties, and functions (including callbacks) from your app. You can delete any of these app elements from code view. From design view, you can delete app components only.

Delete Elements from Code View

To delete a component, property, or function from code view, in the code browser, select the code name, and then press the **Delete** key.

App designer updates the code to remove the deleted element. If you delete a component, App Designer also removes it from design view. When you delete a component, callbacks created for that component are *not* deleted. Therefore, the code you wrote for the callback is preserved.

Tip If you are unsure which name corresponds to a particular component, use the thumbnail view of the app that appears below the code browser. Click the component that you want to identify. App Designer highlights the name of that component in the code browser.

Delete Components from Design View

In the design area, select the component or components that you want to delete, and then press the **Delete** or **Backspace** key.

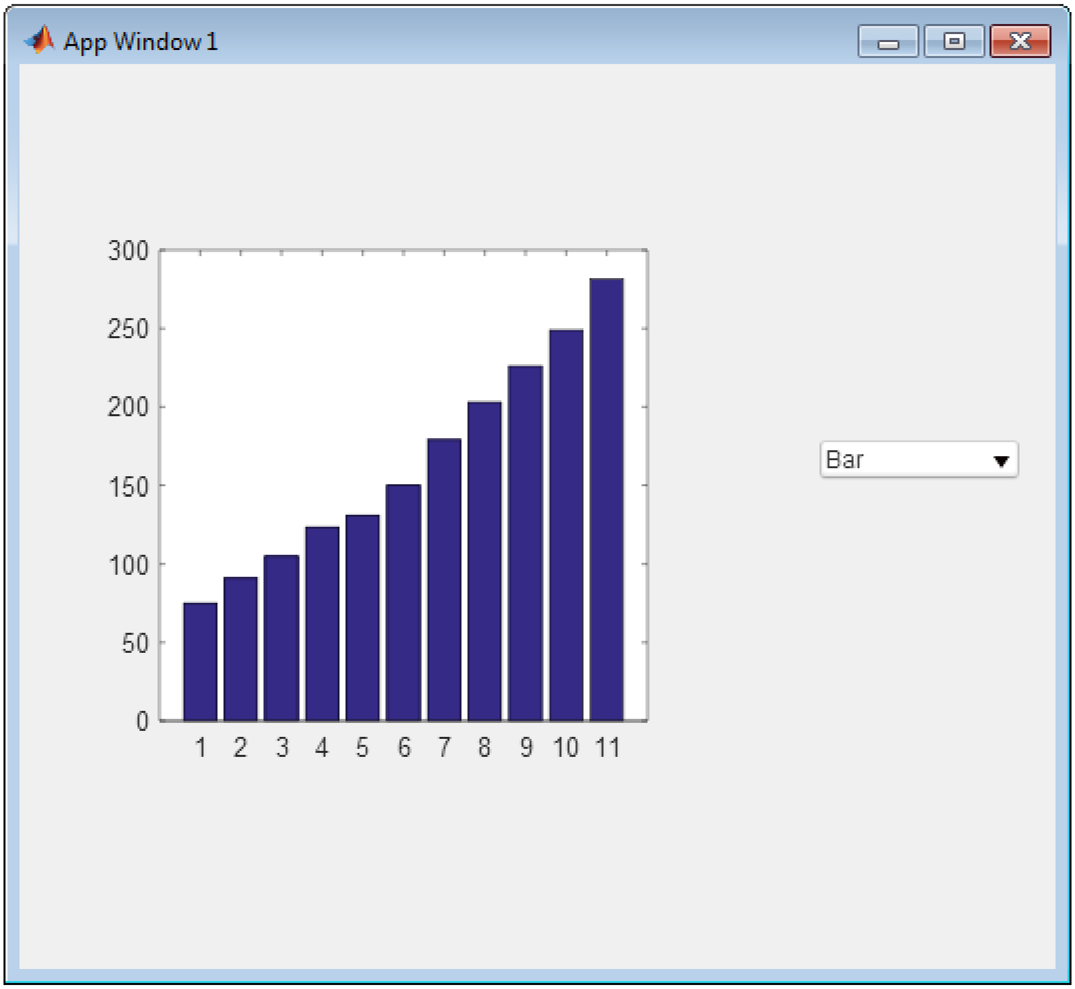
App Designer updates the code and the design area to remove the deleted component. Callbacks created for that component are *not* deleted. Therefore, the code you wrote for the callback is preserved.

App Examples

In this section...

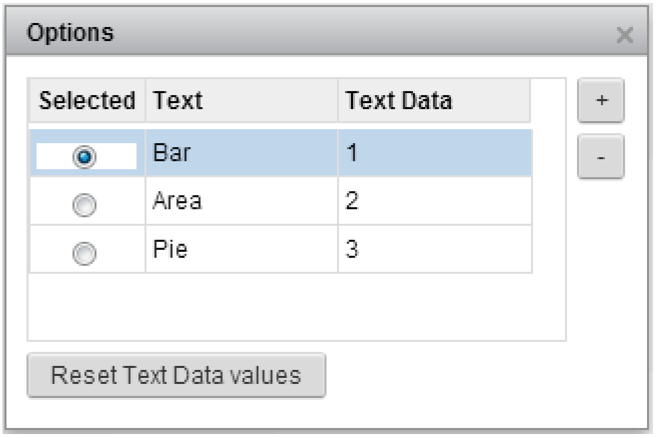
App Displaying Single Plot in an Axes

This example shows how to create an app for users to control the charting method for plotting data. The user selects a charting option from a drop down.



To create the app:

1 In App Designer, drag an axes and a drop down into the central design area. Arrange them as shown in the preceding image.
2 In the design area, select the drop down, and then, on the **Properties** tab, click **Options**.
3 Set the properties for the drop down as shown in the image that follows.



Selected	Text	Text Data
●	Bar	1
○	Area	2
○	Pie	3

To delete a row, select it, and then click the minus button. -.

The user will see the *text* values in the drop down. The *text data* values are numeric values associated with each option. You might find it easier to code callbacks using functions that operate on numeric values rather than string values.

4. Close the Options dialog box, and then click **Code View**.
5. Add an app property to hold plotting data for your app.

   In the code browser, click **Add property**, and then type `Data` in the field that opens.

   App Designer appends the name with the `app.` prefix.
6. Specify the plot that you want to appear in the app when it first opens by adding code to the startup function.

   On the **Editor** tab, click **View Startup Function**. Add this code to the `startup` function:

   ```
   app.Data = [75 91 105 123.5 131 150 179 203 226 249 281.5];
   bar(app.Axes, app.Data);
   ```
7. Create a callback for the drop down.

   In the code browser, right-click **app.DropDown** and select **Callbacks** > **Edit ValueChangedFcn**.

8 In the Add Callback Function dialog box, click **OK**. (Accept the default callback function name.)

9 In the code, beginning on the line after `function DropDownValueChanged(app)`, add the following code:

```
% Get the value data for the option selected by the user
val = app.DropDown.ValueData;
% Plot the data using the selected charting method
    if (val == 1)
      bar(app.Data);
    elseif (val == 2)
      area(app.Data);
    elseif (val == 3)
     pie(app.Data);
    end
```

10 Save and run the app.

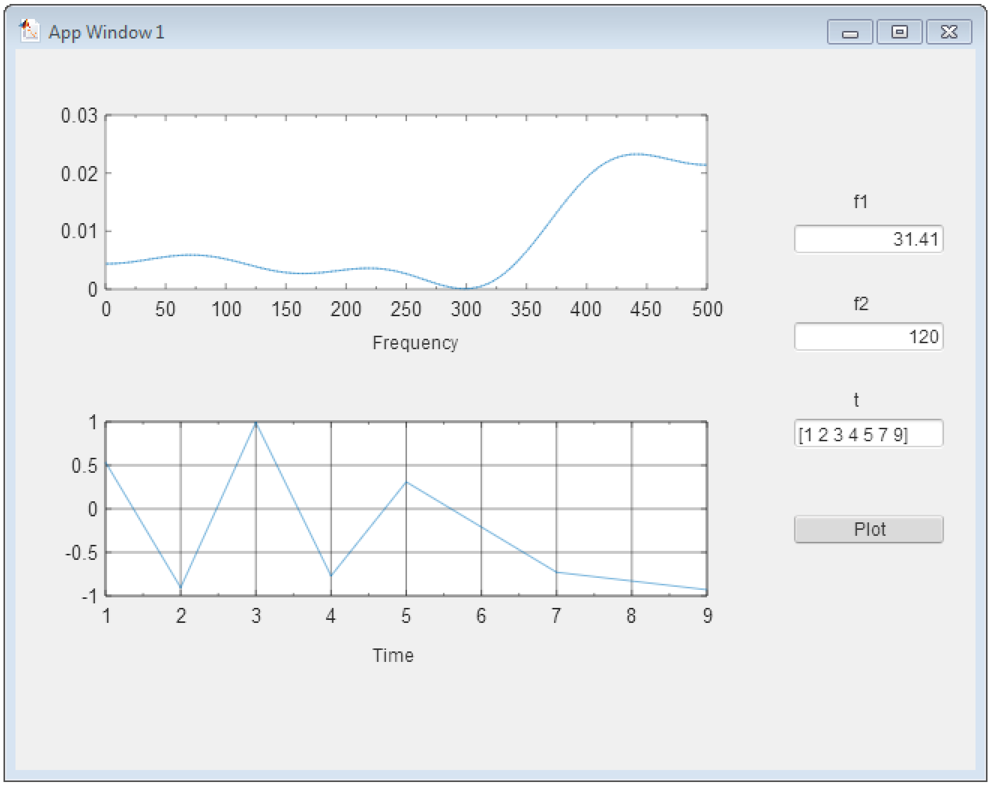
On the **Editor** tab, click **Run App**.

Save the app when prompted.

Test the app by trying the various app controls in the running app.

App Displaying Multiple Axes

This example shows how to create an app that accepts input parameters and plots data in two axes. The parameters define a time-varying and frequency-varying signal. One plot displays the data in the time domain. The other plot displays the data in the frequency domain.



To create the app:

1 In the App Designer, drag the components described in the following table into the central design area. Arrange the components as shown in the preceding image.

Component	Name in Code	Nondefault Property Settings	Label Associated with Component
Axes	AxesFreq	Not Applicable	Frequency
Axes	AxesTime	Not Applicable	Time
Numeric Edit Field	NumericEditFieldf1	**Maximum:** 500	f1
Numeric Edit Field	NumericEditFieldf2	**Maximum:** 500	f2
Edit Field	EditFieldTime	Not Applicable	t

Component	Name in Code	Nondefault Property Settings	Label Associated with Component
Button	Button	**Text:** Plot	Not Applicable

2 Right-click the **Plot** button and select **Callbacks** > **Edit ButtonPushedFcn**.

3 In the Add Callback Function dialog box, click **OK** . (Accept the default callback function name.)

App Designer opens code view.

4 Get the input values, perform calculations, and plot the data in the appropriate axes.

Beginning on the line following `function ButtonButtonPushed(app)` add:

```
% Get user input
f1 = app.NumericEditFieldf1.Value;
f2 = app.NumericEditFieldf2.Value;
t = eval(app.EditFieldTime.Value);

% Calculate data
x = sin(2*pi*f1*t) + sin(2*pi*f2*t);
y = fft(x,512);
m = y.*conj(y)/512;
f = 1000*(0:256)/512;

% Create frequency plot in proper axes
plot(app.AxesFreq,f,m(1:257))
app.AxesFreq.XMinorTick = 'on';
grid on

% Create time plot in proper axes
plot(app.AxesTime,t,x)
app.AxesTime.XMinorTick = 'on';
grid on
```

5 Save and run the app.

On the **Editor** tab, click **Run App**.

Save the app when prompted.

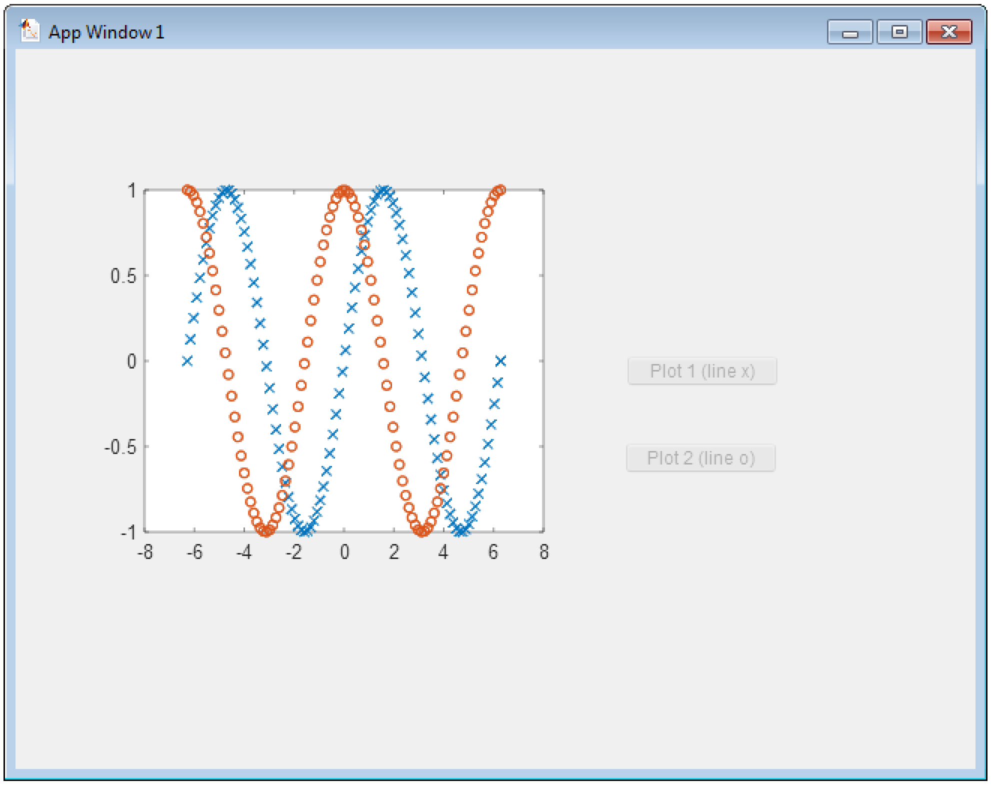
Test the app by trying the various app controls in the running app. Notice that:

- The numeric edit fields, `f1` and `f2`, display an error message if you enter a nonnumeric value or one that exceeds the maximum that you set.

- The edit field for the time vector does not prevent you from entering an invalid vector. To handle invalid time vector values, define a ValueChangedFcn callback function for the `EditFieldTime` component, and then code it to determine if the user's entries are valid.

App Displaying Two Plots in One Axes

This example shows how to create an app that enables the user to control data plotted in an axes using push buttons. The push buttons create different plots in the same axes. After the user presses a button, it is disabled. Because the data being plotted does not change, there is no reason for the user to press each button more than once.



To create the app:

1 In App Designer, drag an axes and two buttons into the central design area. Arrange them as shown in the preceding image and as described in this table.

Component	Text	Name in Code
Button	Plot 1 (line x)	Plot1
Button	Plot 2 (line o)	Plot2

2 Click **Code View**.

3 Add an app property to hold data for your app.

   In the code browser, click **Add property**, and type `X` in the field that opens.

   App Designer appends the name with the `app.` prefix.

4 Specify the value for the x-axis in the startup function.

   On the **Editor** tab, click **View Startup Function**. Add this code to the `startup` function:

```
app.X = linspace(-2*pi,2*pi);
```

5 Create a callback for the Plot1 push button.

   In the code browser, right-click **app.Plot1** and select **Callbacks** > **Edit ButtonPushedFcn**.

6 In the Add Callback Function dialog box, click **OK** . (Accept the default callback function name.)

7 In the code, beginning on the line after `function Plot1ButtonPushed(app)`, add the following code:

```
y = sin(app.X);
plot(app.Axes,app.X,y,'x')
hold(app.Axes,'on');
app.Plot1.Enabled = false;
```

   The last input argument to the `plot` function specifies that the plot will be represented with a series of x characters.

   The `hold` statement ensures that additional plots will not overwrite any existing plots in the specified axes.

8 Create a callback for the Plot2 push button.

In the code browser, right-click **app.Plot2** and select **Callbacks** > **Edit ButtonPushedFcn**.

9 In the Add Callback Function dialog box, click **OK** . (Accept the default callback function name.)

10 In the code, beginning on the line after`function Plot2ButtonPushed(app)`, add the following code:

```
y = cos(app.X);
plot(app.Axes,app.X,y,'o')
hold(app.Axes,'on');
app.Plot2.Enabled = false;
```

11 Save and run the app.

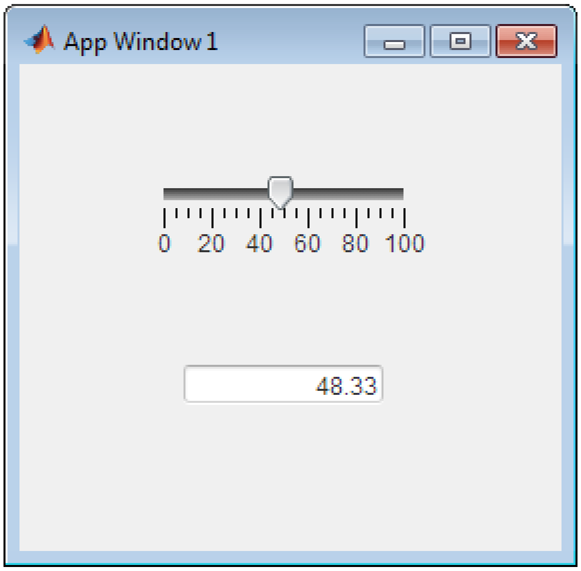
On the **Editor** tab, click **Run App**.

Save the app when prompted.

Test the app by trying the various controls in the running app.

App Displaying Slider Value as it Changes

This example shows how to create an app that displays a slider value in an edit field. The edit field is continuously updated to reflect the current slider value. (Another example, “Create and Run a Simple App” on page 1-3, updates the edit field only when the app user releases the mouse button from the slider.)



To create the app:

1 In App Designer, drag a **Slider** and a **Numeric Edit Field** into the central design area, arranging them as shown in the preceding image.
2 Right-click the slider and select **Callbacks** > **ValueChangingFcn**.
3 In the Add Callback Function dialog box, click **OK**. (Accept the default callback function name.)
4 In the code, beginning on the line after `function SliderValueChanging(app,event)` add this code:

```
app.NumericEditField.Value = event.Value;
```

5 Save and run the app.

   On the **Editor** tab, click **Run App**.

   Save the app when prompted.

Test the app by doing each of the following:

- Drag the slider to a new value.

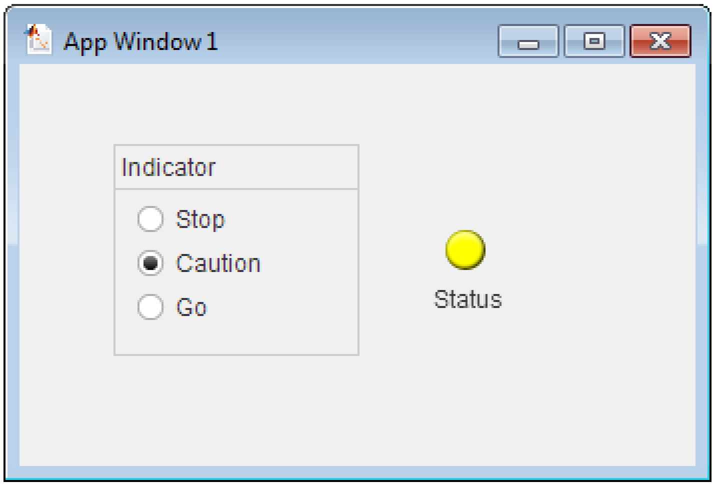
As you do so, the numeric edit field is continuously updated to display the current slider value.

- Click the slider.

  The numeric edit field is updated to display the new slider value.

App to Change Lamp Color Based on Button Group Selection

This example shows how to create an app that changes the color of a lamp based on which radio button the user selects.



To create the app:

1. In App Designer, drag a **Radio Button Group**, a **Lamp**, and a **Label** into the central design area. Arrange them as shown in the preceding image and described in this table.

Component	Title or Text	Name in Code
ButtonGroup	Indicator	ButtonGroup (default)
RadioButton	Stop	StopRadioButton
RadioButton	Caution	CautionRadioButton

Component	Title or Text	Name in Code
RadioButton	Go	GoRadioButton
Lamp	Not applicable	Lamp (default)
Label	Status	Label (default)

2 Select the **Go** radio button.

3 On the **Properties** tab, select the **Selected** check box.

4 Click **Code View**.

5 In the code browser, right click **app.ButtonGroup**, and then select **Callbacks** > **SelectionChangeFcn**.

6 In the Add Callback Function dialog box, click **OK** . (Accept the default callback function name.)

7 Add code to determine which radio button is selected and set the lamp color accordingly.

   In the code, beginning on the line after `function ButtonGroupSelectionChange(app, event)`, add:

```
switch app.ButtonGroup.SelectedObject
    case app.GoRadioButton
        app.Lamp.Color = 'green';
    case app.CautionRadioButton
        app.Lamp.Color = 'yellow';
    case app.StopRadioButton
        app.Lamp.Color = 'red';
end
```

8 Save and run the app.

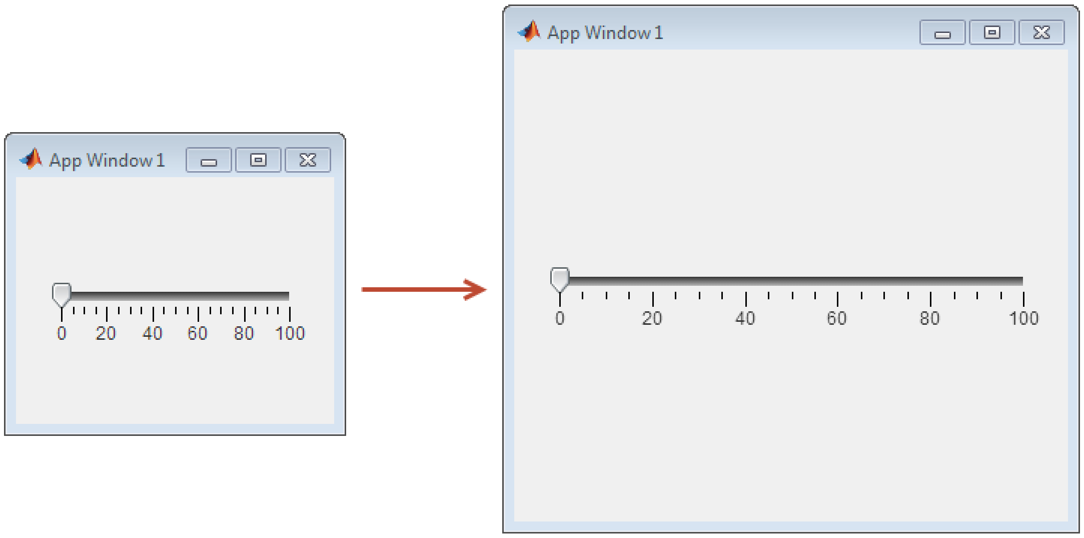
   On the **Editor** tab, click **Run App**.

   Save the app when prompted.

Test the app by selecting each radio button.

App That Resizes Component When Window Resizes

This example shows how to code the callback for app window so that when a user resizes the window, a slider within the window resizes accordingly, remaining centered in the window.



To create the app:

1 In App Designer, drag a **Slider** into the middle of the central design area.

2 Click **Code View**.

3 In the code browser, right-click **app.AppWindow**, and then select **Callbacks** > **Edit SizeChangedFcn**.

4 In the Add Callback Function dialog box, click **OK**. (Accept the default callback function name.)

5 At the cursor location in code view, beginning with the line after `function AppWindowSizeChange(app)` add this code.

```
windowpos = app.AppWindow.Position;
slidersize = app.Slider.Size;
% Subtract the slider height from the
% window height and divide by 2.
app.Slider.Location = [30, (windowpos(4)-6)/2];
% Subtract 60 from the window width.
app.Slider.Size = [windowpos(3)-60, slidersize(2)];
```

The code gets the app window width and height after a resize operation. The code updates the location and size of the slider to keep it centered in the app window, 30 pixels from the left and right sides of the window.

6. Save and run the app.

   On the **Editor** tab, click **Run App**.

   Save the app when prompted.

Test the app by resizing the app window.

Keyboard Shortcuts

Component Selection Shortcuts

Action	Key or Keys
Move the selection to the next component, container, or app window in the design area’s tab key navigation sequence.	**Tab**
Move the selection to the previous component, container, or app window in the design area’s tab key navigation sequence.	**Shift+Tab**
Select all selectable components, excluding those that are children of another component.	**Ctrl+A**
Unselect a previously selected component or reselect an unselected component (toggle selection).	Press **Shift+Click**.

Component Move Shortcuts

Action	Key or Keys
Move selected components down 1 pixel.	**Down Arrow**
Move selected components left 1 pixel.	**Left Arrow**
Move selected components right 1 pixel.	**Right Arrow**
Move selected components up 1 pixel.	**Up Arrow**
Move selected components down 10 pixels.	**Shift+Down Arrow**
Move selected components left 10 pixels.	**Shift+Left Arrow**
Move selected components right 10 pixels.	**Shift+Right Arrow**
Move selected components up 10 pixels.	**Shift+Up Arrow**
Cancel an in-progress operation.	**Escape**

Component Resize Shortcuts

Action	Key
Resize component while maintaining aspect ratio.	Press and hold down the **Shift** key before you begin to drag the component expander.
Cancel an in-progress resize operation.	**Escape**

Component Copy, Paste, Duplicate, and Delete Shortcuts

Action	Key or Keys
Copy selected components to the clipboard.	**Ctrl+C**
Duplicate the selected component (without copying to the clipboard).	**Ctrl+D**
Paste components on the clipboard into the design area.	**Ctrl+V**
Cut selected components (by copying them to the clipboard and removing them from the design area).	**Ctrl+X**
Delete the selected components from the design area.	**Backspace** or **Delete**

Component Alignment Shortcuts

Action	Keys
Align selected components on their left edges.	**Ctrl+1**
Align selected components on their horizontal centers.	**Ctrl+2**
Align selected components on their right edges.	**Ctrl+3**
Align selected components on their top edges.	**Ctrl+4**
Align selected components on their vertical middle.	**Ctrl+5**
Align selected components on their bottom edges.	**Ctrl+6**

Redo, Undo, and Cancel Operation Shortcuts

Action	Key or Keys
Redo an undone design area modification, returning it to the changed state.	**Ctrl+Y** or **Ctrl+Shift+Z**
Redo an undone code editor modification, returning it to the changed state.	**Ctrl+Y**
Undo a design area or code editor modification, returning it to the previous state.	**Ctrl+Z**
Cancel an in-progress resize or move operation.	**Escape**

Run, Save, and Open App Shortcuts

Action	Key or Keys
Save the active app.	**Ctrl+S**
Run the active app.	**F5**
Open a previously saved app.	**Ctrl+O**

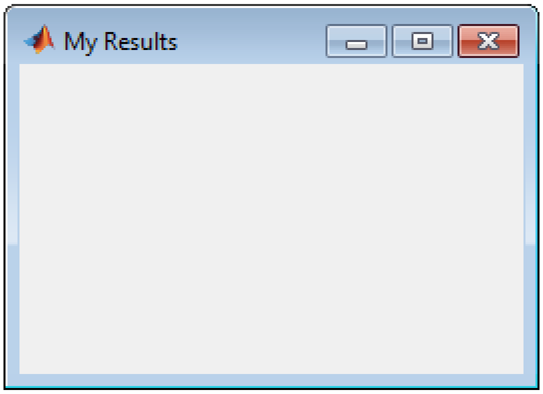
Property Reference

App Window Properties

Control App Window appearance and behavior

An App windows contains an app you create using App Designer. An app window is a figure window tailored for App Designer. The figure properties that are supported and relevant to the App Designer workflow are documented in this section of the documentation. Properties control the appearance and behavior of a particular instance of an app window. To modify aspects of an app window, change property values. Use dot notation to refer to a particular object and property:

```
window = appwindow;
name = window.Name;
window.Name = 'My Results';
```



`Children` — Objects for which App Window is the parent

empty `GraphicsPlaceholder` array (default) | vector of objects

The objects for which App Window is the parent, specified as an array of objects.

To change the stacking of the objects on the display, change their order in the vector.

Example: [o1,o2,o3,o4]

where each `o` is an object.

`Name` — App Window window title

`'App Window'` (default) | string

App Window title, specified as a string.

Example: `'Results'`

`Parent` — App Window parent
root object

App Window parent, returned as a root object.

`Position` — App Window size and location
`[left,bottom,width,height]`

app window size and location on screen (excluding the title bar, menu bar, tool bars, and outer edges), specified as [left, bottom, width, height].

The `left` and `bottom` values define the distance from the lower-left corner of the screen to the lower-left corner of the app window. The `width` and `height` values define the dimensions of the window. The `left` and `bottom` elements can be negative on systems that have more than one monitor. All measurements are in pixels.

Example: `[230,250,570,510]`

`SizeChangedFcn` — Callback function that executes when app window window changes size
function handle | cell array | string

Callback function that executes when the app window size changes, specified as a function handle, cell array containing a function handle and additional arguments, or a string. MATLAB executes this callback routine when any of the following occur:

- A user manually resizes the app window.
- The code resizes the app window.

During execution of the callback function, the handle to the app window being resized is accessible only through the root CallbackObject property, which you can query using `gcbo`.

If you change the app window Position from within the SizeChangedFcn callback, the SizeChangedFcn is not called again.

Example: `@myfun`

Example: `{@myfun,x}`

Example: `{'myfun',x}`

`Units` — Units of measurement

`'pixels'` (default) | `'inches'` | `'centimeters'` | `'normalized'` | `'points'` | `'characters'`

Units of measurement, specified as `'pixels'`, `'inches'`, `'centimeters'`, `'normalized'`, `'points'`, or `'characters'`.

Note: Regardless of the Units property value that you specify, MATLAB always uses pixel units.

`Visible` — App Window visibility

`'on'` (default) | `'off'`

App Window visibility, specified as `'on'` or `'off'`. The Visible property determines whether the app window is displayed on the screen. If the Visible property of an app window is set to `'off'`, the entire app window is hidden, but you can still specify and access its properties.

Changing the size of a hidden app window triggers the `SizeChangedFcn` callback if and when the app window becomes visible.

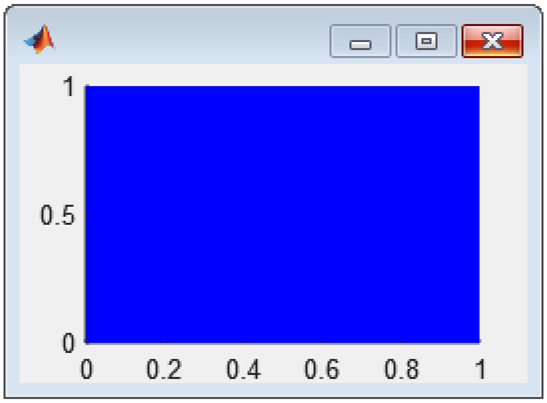
Note Changing the Visible property of an app window does *not* change the values of the Visible properties of its child components, even though hiding the app window causes the components to be hidden.

Axes Properties

Control axes appearance and behavior

Axes properties control the appearance and behavior of an axes object. By changing property values, you can modify certain aspects of the axes. Use dot notation to refer to a particular object and property:

```
window = appwindow;
ax = axes('Parent',window);
color = ax.Color;
ax.Color = 'blue';
```



`ActivePositionProperty` — Position property to hold constant during resize operation
`'outerposition'` (default) | `'position'`

Position property to hold constant during a resize operation, specified as one of these values:

- `'outerposition'` — Hold the `OuterPosition` property constant. If you resize the app window, the window layout adjusts to accommodate the the xlabel, ylabel and title strings.
- `'position'` — Hold the `Position` property constant. If you resize the app window, the window layout does not adjust to accommodate the xlabel, ylabel and title strings.

An app window can change size either interactively or during a printing or exporting operation.

`BeingDeleted` — Deletion status of axes
`'off'` (default) | `'on'`

Deletion status of axes, returned as `'on'` or `'off'`. MATLAB sets the BeingDeleted property to `'on'` when the delete function of the axes begins execution (see the DeleteFcn property). The BeingDeleted property remains set to `'on'` until the axes no longer exists.

Check the value of the BeingDeleted property to verify that the axes is not about to be deleted before querying or modifying the axes properties.

`Box` — Axes box outline
`'off'` (default) | `'on'`

Axes box outline, specified as one of these values:

- `'off'` — Does not display the box outline around the axes.
- `'on'` — Displays the box outline around the axes.

The XColor and YColor properties control the color of the outline.

`Children` — Children of axes object
empty `GraphicsPlaceholder` array (default) | array of graphics objects

Children of axes object, returned as an array of graphics objects. You cannot add or remove children using the `Children` property of the axes.

To add a child to this list, set the `Parent` property of the child graphics object to the axes object.

`CLim` — Color mapping for objects using colormap
`[0 1]` (default) | `[cmin cmax]`

Color mapping for objects in axes that use the colormap, specified as a two-element vector of the form `[cmin cmax]`.

The CLim property determines how MATLAB maps data values to the colors in the colormap, where:

- `cmin` specifies the data value that maps to the first color in the colormap.
- `cmax` specifies the data value that maps to the last color in the colormap.

The axes linearly interpolates data values between `cmin` and `cmax` across the colormap. Values outside this range use either the first or last colormap color, whichever is closest.

For information on changing the colormap, see the `colormap` function. For information on color mapping, see the `caxis` function.

Setting this property sets the CLimMode property to `'manual'`.

`CLimMode` — Selection mode for `CLim`

`'auto'` (default) | `'manual'`

Selection mode for the CLim property, specified as one of these values:

- `'auto'` — Set the CLim property to span the CData limits of the graphics objects displayed in the axes.
- `'manual'` — Use the manually specified CLim values. To specify the values, set the CLim property. The axes does not change the value of the `CLim` property when the limits of axes children change.

`Color` — Color of axes back planes

`[1 1 1]` (default) | RGB triplet | color string | `'none'`

Color of axes back planes, specified as an RGB triplet, a color string, or `'none'`. If you set the color to `'none'`, then the axes is invisible and the app window color shows through.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Example: `[0 0 1]`

Example: `'b'`

Example: `'blue'`

`ColorOrder` — Colors for multiline plots

seven predefined colors (default) | three-column matrix of RGB triplets

Colors for multiline plots, specified as a three-column matrix of RGB triplets, where each row defines one color in the color order. The functions that create line plots cycle through the colors defined by the `ColorOrder` property to color each line plotted.

The default color order has seven colors.

Default Color Order	Associated RGB Triplets

🟦🟧🟨🟪🟩🟦🟥	`[     0   0.4470   0.7410`
	`0.8500   0.3250   0.0980`
	`0.9290   0.6940   0.1250`
	`0.4940   0.1840   0.5560`
	`0.4660   0.6740   0.1880`
	`0.3010   0.7450   0.9330`
	`0.6350   0.0780   0.1840]`

The functions that create line plots cycle through the colors to color each line. The hold state for the axes affects the colors used:

- If the hold state for the axes is off (when the NextPlot property is set to `'replace'`, then high-level plotting functions such as `plot` reset the color order to the default colors before plotting. If you want to specify new colors for the color order and do not want high-level plotting functions to reset them, then set the NextPlot property to `'replacechildren'`. Alternatively, you can specify a new default value for the ColorOrder property on the root using the `set` function. For example, `set(groot,'defaultAxesColorOrder',[0 1 0; 0 0 1])`.
- If the hold state from the axes is on (when the NextPlot property is set to `'add'`), then plotting functions cycle through the colors starting from the place in the color order where the last plot ended.

Data Types: `single` | `double`

`CreateFcn` — Creation callback

`''` (default) | function handle | cell array | string

Creation callback, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB command or function, which is evaluated in the base workspace (not recommended)

Use this property to execute code when the axes is created. Setting the `CreateFcn` property on an existing axes has no effect. You must define a default value for this property, or define this property using a `Name,Value` pair during axes creation. MATLAB executes the callback after creating the axes and setting all of its properties.

If you specify this callback using a function handle, MATLAB passes two arguments to the callback function when executing the callback:

- The axes object — You can access properties of the axes object from within the callback function. You also can access the axes object through the `CallbackObject` property of the root, which can be queried using the `gcbo` function.
- Event data — This argument is empty for this property. Replace it with the tilde character (~) in the function definition to indicate that this argument is not used.

For more information on how to use function handles to define callback functions, see the "Callback Definition" topic in the MATLAB documentation.

Example: `@myCallback`

Example: `{@myCallback,arg3}`

`DataAspectRatio` — Relative length of data units along each axis

[1 1 1] (default) | three-element vector of the form `[dx dy n]`

Relative length of data units along each axis, specified as a three-element vector of the form `[dx dy n]`. The first two vector elements define the relative x and y data scale factors. The third vector element is ignored, but must be included. For example, specifying this property as `[1 2 1]` sets the length of one unit of data in the *x*-direction to be the same length as two units of data in the *y*-direction.

The DataAspectRatio property interacts with the `PlotBoxAspectRatio`, `XLimMode`, and `YLimMode` properties to control how MATLAB scales the x-axis and y-axis.

Setting a value for the DataAspectRatio disables the "stretch-to-fill the app window shape behavior" if DataAspectRatioMode and PlotBoxAspectRatioMode are both set to `'auto'`.

If the associated mode property is set to `'auto'`, then MATLAB chooses the ratio values. If you set this property, then MATLAB sets the mode to `'manual'`.

Example: `[1 1 1]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DataAspectRatioMode` — Selection mode for `DataAspectRatio`
`'auto'` (default) | `'manual'`

Selection mode for DataAspectRatio specified as one of these values:

- `'auto'` — Use values that make best use of the area provided by the app window.
- `'manual'` — Use the manually specified values. To specify the values, set the DataAspectRatioMode to `'manual'`. Changing DataAspectRatioMode to `'manual'` disables the "stretch-to-fill the app window shape" behavior if DataAspectRatioMode and PlotBoxAspectRatioMode property values are both set to `'auto'`.

This table describes the interactions among properties when you disable the stretch-to-fill behavior.

XLimitMode, and YLimitMode	DataAspectRatioMode	PlotBoxAspectRatioMode	Behavior
`'auto'`	`'auto'`	`'auto'`	Limits chosen to span data range in all dimensions.
`'auto'`	`'auto'`	`'manual'`	Limits chosen to span data range in all dimensions. MATLAB modifies DataAspectRatio to achieve the requested PlotBoxAspectRatio within the limits the software selected.
`'auto'`	`'manual'`	`'auto'`	Limits chosen to span data range in all dimensions. MATLAB modifies

XLimitMode, and YLimitMode	DataAspectRatioMode	PlotBoxAspectRatioMode	Behavior
			PlotBoxAspectRatio to achieve the requested DataAspectRatio within the limits the software selected.
`'auto'`	`'manual'`	`'manual'`	Limits chosen to completely fit and center the plot within the requested PlotBoxAspectRatio given the requested DataAspectRatio (this might produce empty space around 2 of the 3 dimensions).
`'manual'`	`'auto'`	`'auto'`	MATLAB honors limits and modifies the DataAspectRatio and PlotBoxAspectRatio as necessary.
`'manual'`	`'auto'`	`'manual`	MATLAB honors limits and PlotBoxAspectRatio and modifies DataAspectRatio as necessary.
`'manual'`	`'manual'`	`'auto'`	MATLAB honors limits and DataAspectRatio and modifies the PlotBoxAspectRatio as necessary.

`DeleteFcn` — Deletion callback
`''` (default) | function handle | cell array | string

Deletion callback, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB command or function, which is evaluated in the base workspace (not recommended)

Use this property to execute code when the axes is deleted. MATLAB executes the callback before destroying the axes so that the callback can access its property values.

If you specify this callback using a function handle, MATLAB passes two arguments to the callback function when executing the callback:

- The axes object — You can access properties of the axes object from within the callback function. You also can access the axes object through the `CallbackObject` property of the root, which can be queried using the `gcbo` function.
- Event data — This argument is empty for this property. Replace it with the tilde character (~) in the function definition to indicate that this argument is not used.

For more information on how to use function handles to define callback functions, see the "Callback Definition" topic in the MATLAB documentation.

Example: `@myCallback`

Example: `{@myCallback,arg3}`

`FontAngle` — Character slant

`'normal'` (default) | `'italic'`

Character slant, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system. Not all fonts have both font styles, therefore, the italic font might look the same as the normal font.

`FontSize` — Font size

`10` (default) | scalar numeric value

Font size, specified as a scalar numeric value. The FontSize property determines the size of the text used for the axis labels and the title. The FontUnits property determines the units used to interpret the font size.

Example: `12`

`FontUnits` — Font size units

`'points'` (default) | `'inches'` | `'centimeters'` | `'characters'` | `'normalized'` | `'pixels'`

Font size units, specified as one of the values in the table that follows.

Units	Description
`'points'`	Points. One point equals `1/72` inches.
`'inches'`	Inches
`'centimeters'`	Centimeters
`'characters'`	Based on the size of characters in the default system font. The width of one character unit is the width of the letter `x`. The height of one character unit is the distance between the baselines of two lines of text.
`'normalized'`	Interpreted as a fraction of the axes height. If you resize the axes, MATLAB modifies the font size accordingly. For example, if the `FontSize` is `0.1` in normalized units, then the text is 1/10 of the axes height.
`'pixels'`	Pixels. Pixel size depends on the screen resolution.

If you set the FontSize and the FontUnits in one function call, you must set the FontUnits property first so that the axes correctly interprets the specified FontSize.

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`GridColor` — Color of grid lines

`[0.15 0.15 0.15]` (default) | RGB triplet | color string | `'none'`

Color of grid lines, specified as an RGB triplet, a color string, or `'none'`.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

To set the colors for the axes box outline, use the XColor and YColor properties.

Example: `[0 0 1]`

Example: `'b'`

Example: `'blue'`

`GridColorMode` — Selection mode for `GridColor`

`'auto'` (default) | `'manual'`

Selection mode for the GridColor property, specified as `'auto'` or `'manual'`. The color is based on the values of the GridColorMode, `XColorMode`, and `YColorMode` properties.

These table lists the grid line colors for different combinations of color modes.

GridColorMode Property	XColorMode Property	x-Axis Grid Line Color
`'auto'`	`'auto'`	Determined by `GridColor` property
`'auto'`	`'manual'`	Determine by `XGridColor`property

GridColorMode Property	XColorMode Property	x-Axis Grid Line Color
`'manual'`	`'auto'`	Determined by `GridColor` property
`'manual'`	`'manual'`	Determined by `GridColor` property

GridColorMode Property	YColorMode Property	y-Axis Grid Line Color
`'auto'`	`'auto'`	Determined by `GridColor` property
`'auto'`	`'manual'`	Determine by `YGridColor`property
`'manual'`	`'auto'`	Determined by `GridColor` property
`'manual'`	`'manual'`	Determined by `GridColor` property

`LineStyleOrder` — Line styles and markers for multiline plots
`'-'` solid line (default) | cell array of specifiers

Line styles and markers for multiline plots, specified as a cell array of any number and combination of the specifiers in this table.

Specifier	Line Style
`'-'`(default)	Solid line
`'+'`	Plus sign markers
`'o'`	Circle markers
`'*'`	Star markers
`'.'`	Point markers
`'x'`	Cross markers
`'s'`	Square markers
`'d'`	Diamond markers
`'^'`	Upward-pointing triangle markers
`'v'`	Downward-pointing triangle markers
`'>'`	Right-pointing triangle markers

Specifier	Line Style
`'<'`	Left-pointing triangle markers
`'p'`	Five-pointed star (pentagram) markers
`'h'`	Six-pointed star (hexagram) markers

Combine strings to specify lines with markers, such as `'-*'`. The plot cycles through the line styles only after using all the colors contained in the ColorOrder property. If the hold state for the axes is off (when the NextPlot property of the axes is set to `'replace'`), then high-level plotting functions such as `plot` reset the line style order to the default line styles before plotting.

If you want to specify a set of line styles that is different from the default and do not want high-level plotting functions to reset the `LineStyleOrder` property, then set NextPlot to `'replacechildren'`. Alternatively, you can specify a new default value for the LineStyleOrder property on the root using the `set` function. For example, `set(groot,'defaultAxesLineStyleOrder',{'-*',':','o'})`.

Example: `{'-*',':','o'}`

`LineWidth` — Width of axes outline, tick marks, and grid lines

`0.5` (default) | scalar value

Width of axes outline, tick marks, and grid lines, specified as a scalar value in point units. One point equals `1/72` inch. The default value is `0.5` point.

Example: `1.5`

`MinorGridColor` — Color of minor grid lines

`[0.1 0.1 0.1]` (default) | RGB triplet | color string

Color of minor grid lines, specified as an RGB triplet, a color string, or `'none'`.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`

Long Name	Short Name	RGB Triplet
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Example: `[0 0 1]`

Example: `'b'`

Example: `'blue'`

`NextPlot` — Properties to reset when adding new plot

`'replace'` (default) | `'add'` | `'replacechildren'`

Properties to reset when adding a new plot to the axes, specified as one of these values:

- `'replace'` — Reset all axes properties, except Position and Units, to their default values and delete all axes children before displaying the new plot. Using this value is similar to using `cla reset`.
- `'add'` — Use the existing axes to add more plots to the axes and do not reset properties.
- `'replacechildren'` — Remove all child objects before adding new plots, but do not reset axes properties. This is similar to using `cla`.

The `newplot` function simplifies the use of the NextPlot property and is useful for writing custom graphing functions.

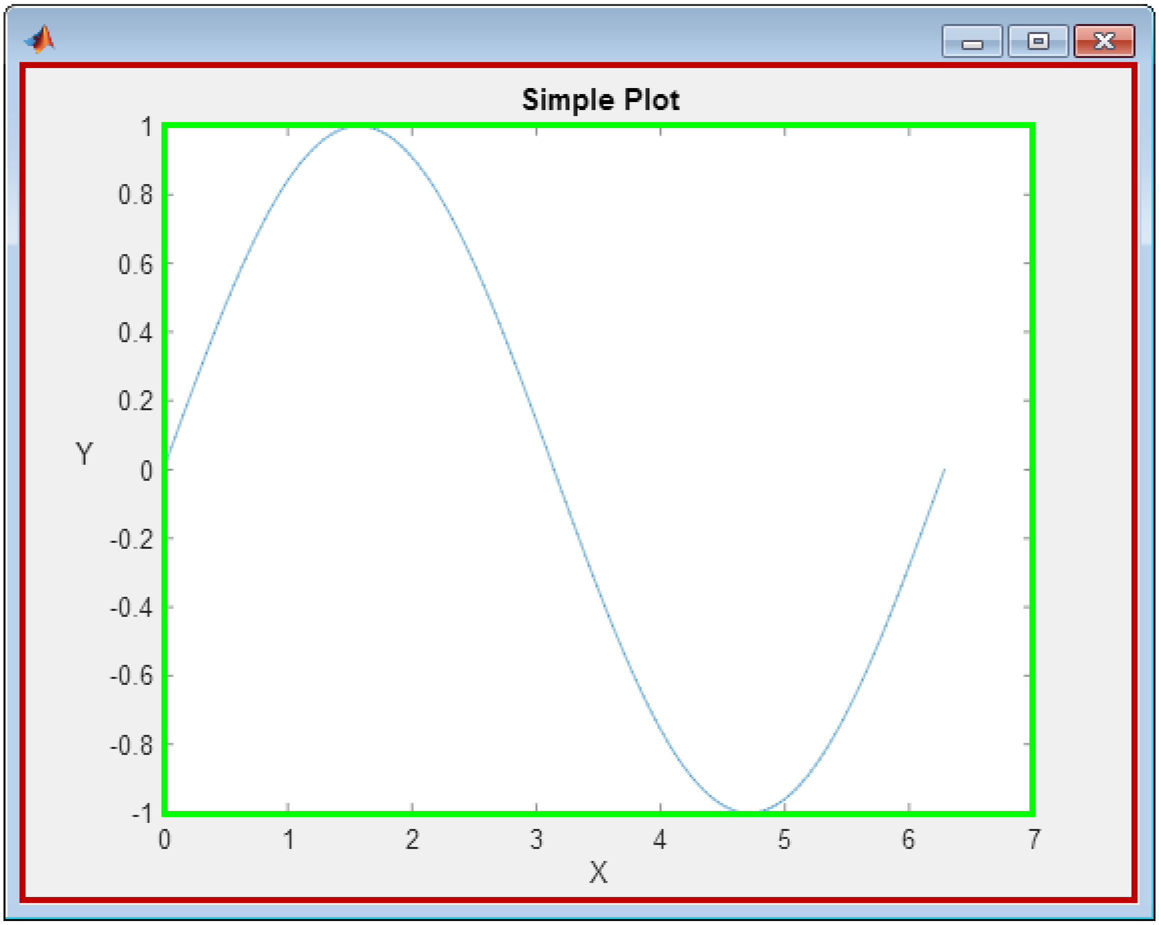
`OuterPosition` — Size and location of axes including labels, title, and margins

`[0 0 1 1]` (default) | `[left bottom width height]`

Size and location of axes including labels, title, and margins, specified as a four-element vector of the form `[left bottom width height]`. This vector defines the extents of the rectangle that encloses the outer bounds of the axes. The `left` and `bottom` elements define the distance from the lower-left corner of the app window that contains the axes to the lower-left corner of the rectangle. The last two elements, `width` and `height` elements are the rectangle dimensions.

The values are measured in units normalized to the container. The default value of `[0 0 1 1]` includes the whole interior of the container.

In the following image, the red rectangle encloses the region defined by the OuterPosition property. The green rectangle encloses the region defined by the Position property.



For more information, see the “Axes Resize to Accommodate Titles and Labels” topic in the MATLAB documentation.

`Parent` — Parent of axes
app window object

Parent of axes, specified an app window object.

`PlotBoxAspectRatio` — Relative length of each axis

`[1 1 1]` | `[px py n]`

Relative length of each axis, specified as a three-element vector of the form `[px py n]` defining the relative x-axis and y-axis scale factors. The third vector element is ignored, but must be present. The plot box is a box enclosing the axes data region as defined by the axis limits.

The PlotBoxAspectRatio property interacts with the DataAspectRatio, XLimMode and YLimMode properties. Setting the PlotBoxAspectRatio disables the “stretch-to-fill the app window shape’ behavior, if DataAspectRatioMode and PlotBoxAspectRatioMode property values are both `'auto'`.

If the associated mode property is set to `'auto'`, then MATLAB chooses the ratio values. If you set this property, then MATLAB sets the mode to `'manual'`.

Example: `[1,0.75,0.75]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`PlotBoxAspectRatioMode` — Selection mode for `PlotBoxAspectRatio`

`'auto'` (default) | `'manual'`

Selection mode for the PlotBoxAspectRatio property, specified as one of these values:

- `'auto'` — Use values that make best use of the area provided by the app window.
- `'manual'` — Use the manually specified values. To specify the values, set the PlotBoxAspectRatio property. Changing PlotBoxAspectRatioMode to `'manual'` disables the “stretch-to-fill the app window shape” behavior if PlotBoxAspectRatioMode and DataAspectRatioMode property values are both `'auto'`.

This table describes the behavior for various combinations of properties when you disable the stretch-to-fill behavior.

XLimitMode, and YLimitMode	DataAspectRatioMode	PlotBoxAspectRatioMode	Behavior
`'auto'`	`'auto'`	`'auto'`	Limits chosen to span data range in all dimensions.

XLimitMode, and YLimitMode	DataAspectRatioMode	PlotBoxAspectRatioMode	Behavior
`'auto'`	`'auto'`	`'manual'`	Limits chosen to span data range in all dimensions. MATLAB modifies DataAspectRatio to achieve the requested PlotBoxAspectRatio within the limits the software selected.
`'auto'`	`'manual'`	`'auto'`	Limits chosen to span data range in all dimensions. MATLAB modifies PlotBoxAspectRatio to achieve the requested DataAspectRatio within the limits the software selected.
`'auto'`	`'manual'`	`'manual'`	Limits chosen to completely fit and center the plot within the requested PlotBoxAspectRatio given the requested DataAspectRatio (this might produce empty space around 2 of the 3 dimensions).
`'manual'`	`'auto'`	`'auto'`	MATLAB honors limits and modifies the DataAspectRatio and PlotBoxAspectRatio as necessary.

XLimitMode, and YLimitMode	DataAspectRatioMode	PlotBoxAspectRatioMode	Behavior
`'manual'`	`'auto'`	`'manual`	MATLAB honors limits and PlotBoxAspectRatio and modifies DataAspectRatio as necessary.
`'manual'`	`'manual'`	`'auto'`	MATLAB honors limits and DataAspectRatio and modifies the PlotBoxAspectRatio as necessary.

`Position` — Size and position of axes within app window
`[0.1300 0.1100 0.7750 0.8150]` (default) | `[left bottom width height]`

Size and position of the axes within the app window that contains the axes, specified as a four-element vector of the form `[left bottom width height]`. The `left` and `bottom` elements define the distance from the lower-left corner of the app window to the lower-left corner of the axes. The `width` and `height` elements are the axes dimensions.

The values are measured in units normalized to the app window.

The axes dimensions are the largest possible values that conform to all other properties and do not extend outside the `Position` rectangle. Axes properties that affect the axes size and shape include the `DataAspectRatio` and `PlotBoxAspectRatio`.

Example: `[0 0 1 1]`

`TickDir` — Tick mark direction
`'in'` (default) | `'out'`

Tick mark direction, specified as one of these values:

- `'in'` — Directs tick marks inward from the axis lines.
- `'out'` — Directs tick marks outward from the axis lines.

Setting this property sets the TickDirMode property to `'manual'`.

`TickDirMode` — Selection mode for `TickDir`
`'auto'` (default) | `'manual'`

Selection mode for the TickDir property, specified as one of these values:

- `'auto'` — Use default tick direction
- `'manual'` — Use manually specified tick mark direction. To specify the tick mark direction, set the TickDir property.

`TickLength` — Tick mark length

`[0.01 0.025]` (default) | `[length,n]`

Tick mark length, specified as a two-element vector of the form `[length, n]`. The first element is the tick mark length. The second element is ignored (but must be specified). Specify the values in units normalized relative to the longest of the visible *x*-axis or *y*-axis lines.

Example: `[0.02 0.035]`

`Title` — Text object used for axes title

text object (default)

Text object used for axes title. Use this text object to change properties of the title text.

```
window = appwindow;
ax = axes('Parent',window);
ax.Title.String = 'My Graph Title';
ax.Title.FontWeight = 'normal';
```

Note: To access the axes title text object, use the Title property or the `title` function. This text object is not contained in the axes `Children` property, it cannot be returned by `findobj`, and it does not use default values defined for text objects.

`Type` — Type of graphics object

`'axes'` (default)

Type of graphics object returned as the string `'axes'`.

Data Types: `char`

`Units` — Position units

`'normalized'` (default) | `'inches'` | `'centimeters'` | `'pixels'` | `'points'` | `'characters'`

Position units, specified as one of the values in the table that follows.

`Units`	**Description**
`'normalized'` (default)	Normalized with respect to the container, which is typically the figure or a uipanel. The lower-left corner of the conatiner maps to `(0,0)` and the upper-right corner maps to `(1,1)`.
`'inches'`	Inches.
`'centimeters'`	Centimeters.
`'characters'`	Based on the size of characters in the default system font. The width of one character unit is the width of the letter `x`. The height of one character unit is the distance between the baselines of two lines of text.
`'points'`	Points. One point equals `1/72` inch.
`'pixels'`	Pixels. Pixel size depends on the screen resolution.

When specifying the units as a `Name,Value` pair during object creation, you must set the `Units` property before specifying the properties that you want to use these units, such as `Position`.

`Visible` — Visibility of axes

`'on'` (default) | `'off'`

Visibility of axes, specified as one of these values:

- `'on'` — Display the axes.
- `'off'` — Hide the axes without deleting it. You still can access the properties of a invisible axes object.

`XAxisLocation` — Location of x-axis

`'bottom'` (default) | `'top'`

Location of the *x*-axis, specified as one of these values:

- `'bottom'` — Display x-axis at bottom of axes

- or `'top'` — Display x-axis at top of axes

`XColor`, `YColor` — Color of axes outline and tick marks

`[0.15 0.15 0.15]` (default) | RGB triplet | color string | `'none'`

Color of the axes outline in the *x* or *y* direction and the associated tick marks, specified as an RGB triplet, a color string, or `'none'`. If you set the color to `'none'`, then the axes outline is invisible.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Example: `[1 1 0]`

Example: `'y'`

Example: `'yellow'`

`XColorMode`, `YColorMode` — Selection mode for axes outline color

`'auto'` (default) | `'manual'`

Selection mode for the axis outline color, specified as one of these values:

- `'auto'` — Use the default color.
- `'manual'` — Use the manually specified color. To specify the color, set the XColor or YColor property.

See `GridColorMode` for related information about grid line color selection.

`XDir`, `YDir` — Direction of increasing values along axis

`'normal'` (default) | `'reverse'`

Direction of increasing values along axis, specified as one of these values:

- `'normal'` — Normal direction of increasing values:
  - x-axis values increase from left to right.
  - y-axis values increase from bottom to top.
- `'reverse'` — Reverse direction of increasing values:
  - x-axis values decrease from left to right.
  - y-axis values decrease from bottom to top.

`XGrid`, `YGrid` — Display of grid lines

`'off'` (default) | `'on'`

Display of grid lines, specified as one of these values:

- `'off'` — Do not display the grid lines.
- `'on'` — Displays grid lines perpendicular to the axis, for example, along lines of constant `x` or `y` values. Use the `grid` command to set both properties `'on'` or `'off'`.

`XLabel`, `YLabel` — Text object

text object (default)

Text object for *x*-axis or *y*-axis, label. Use this text object to change the properties of the axis label text.

```
window = appwindow;
ax = axes('Parent',window);
ax.YLabel.String = 'Y Axis';
ax.YLabel.FontSize = 12;
```

Note: To access the axis label text objects, use XLabel and YLabel, properties or the `xlabel` and `ylabel` functions. These text objects are not contained in the axes `Children` property, they cannot be returned by `findobj`, and they do not use default values defined for text objects.

`XLim`, `YLim` — Minimum and maximum axis limits

`[0 1]` (default) | `[min max]`

Minimum and maximum *x*-axis or *y*-axis limits, specified as a two-element vector of the form `[min max]`.

If the associated mode property is set to `'auto'`, then MATLAB chooses the axis limits. If you assign a value to this property, then MATLAB sets the mode to `'manual'` and does not automatically compute the limits.

`XLimMode`, `YLimMode` — Selection mode for axis limits

`'auto'` (default) | `'manual'`

Selection mode for axis limits, specified as one of these values:

- `'auto'` — Select axis limits based on the data plotted. That is, the total span of the XData or YData of all the objects displayed in the axes.
- `'manual'` — Use manually specified axis limits. To specify the axis limits, set the XLim or YLim property.

`XMinorGrid`, `YMinorGrid` — Display of minor grid lines

`'off'` (default) | `'on'`

Display of minor grid lines, specified as one of these values:

- `'off'` — Do not display grid lines.
- `'on'` — Display grid lines aligned with the minor tick marks of the axis. You do not need to enable minor ticks to display minor grid lines.

`XMinorTick`, `YMinorTick` — Display of minor tick marks

`'off'` (default) | `'on'`

Display of minor tick marks, specified as one of these values:

- `'off'` — Do not display minor tick marks.
- `'on'` — Display minor tick marks between the major tick marks on the axis. The space between the major tick marks determines the number of minor tick marks.

`XScale`, `YScale` — Scale of values along axis

`'linear'` (default) | `'log'`

Scale of values along axis, specified as `'linear'` or `'log'`.

`XTick`, `YTick` — Tick mark locations

`[ ]` (default) | vector of increasing values

Tick mark locations, specified as a vector of increasing values.

If the associated mode property is set to `'auto'`, then MATLAB chooses the tick values. If you assign a value to this property, then MATLAB sets the mode to `'manual'` and does not automatically recompute the tick values.

Example: `[2 4 6 8 10]`

Example: `0:10:100`

Data Types: `single` | `double`

`XTickLabel`, `YTickLabel` — Tick mark labels

`''` (default) | cell array of strings

Tick mark labels, specified as a cell array of strings. If you do not specify enough strings for all the ticks marks, then the axes cycles through the specified strings.

If the associated mode property is set to `'auto'`, then MATLAB chooses the labels. If you assign a value to this property, then MATLAB sets the mode to `'manual'` and does not automatically choose the labels.

`XTickLabelMode`, `YTickLabelMode` — Selection mode for tick mark labels

`'auto'` (default) | `'manual'`

Selection mode for tick mark labels, specified as one of these values:

- `'auto'` — Label tick marks with numeric values that span the range of the plotted data.
- `'manual'` — Use manually specified tick mark labels. To specify the labels, set the XTickLabel or YTickLabel property.

`XTickMode`, `YTickMode` — Selection mode for tick mark locations

`'auto'` (default) | `'manual'`

Selection mode for tick mark locations, specified as one of these values:

- `'auto'` — Select the tick mark locations based on the range of data for the axis.
- `'manual'` — Use manually specified tick mark locations. To specify the values, set the XTick or YTick property.

`YAxisLocation` — Location of y-axis

`'left'` (default) | `'right'`

Location of *y*-axis, specified as one of these values:

- `'left'` — Display y-axis on right side of axes.

  \

  `'right'` — Display y-axis on left side of axes.

See Also

`axes` | `axis` | `box` | `caxis` | `cla` | `gca` | `grid`

More About

- “Access Property Values”

Button Properties

Control push button appearance and behavior

Buttons are app components that typically generate an action when an app user presses and releases them. Button properties control the appearance and behavior of a particular instance of a button. To modify aspects of a button, change property values. Use dot notation to refer to a particular object and property:

```
button = uibutton;
txt = button.Text;
button.Text = 'Plot';
```

`ButtonPushedFcn` — Code to execute after app user clicks and releases button

[ ] (default) | function handle | cell array | string

Code to execute when the app user clicks and releases the button, specified as a function handle, a cell array containing function handle and additional arguments, or a string.

MATLAB does not execute the callback function if the app user clicks the button, but then moves the mouse and releases the mouse button while the cursor is outside the bounds of the button.

Example: @function

Example: {@function, x}

`Enabled` — Operational state of Button

`true` (default) | `false`

Operational state of button, specified as `true` or `false`.

If you set this property to `true`, the app user can press the button.

If you set this property to `false`, the button appears dimmed, indicating that an app user cannot press it and the button will not trigger a callback.

`FontAngle` — Character slant of button text

`'normal'` (default) | `'italic'`

Character slant of the button text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display button `text`
`'Helvetica'` (default) | string

Font in which to display the button text, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

Example: `'Arial'`

`FontSize` — Font size of button text
12 (default) | positive number

Font size of button text, specified as a positive number. The units are points.

Example: 14

Data Types: `double`

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

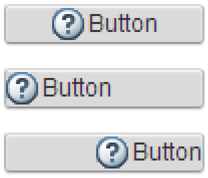
- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`HorizontalAlignment` — Horizontal alignment of icon and text within button
`'center'` (default) | `'left'` | `'right'`

Horizontal alignment of text and icon within the button, specified as `'center'`, `'left'`, or `'right'`.

The following image shows the three horizontal alignment options, center, left, and right, when the VerticalAlignment and IconAlignment properties are set to their default values (`'center'` and `'left'`, respectively).



`Icon` — File name of icon to display on button
`''` (default) | string

File name of icon to display on button, specified as a string.

The string must be the name and extension of an image file on the MATLAB path, or the full path to an image file.

The image file type must be JPEG, GIF, or PNG.

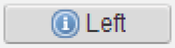
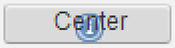
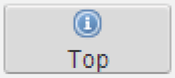
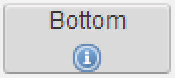
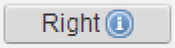
MATLAB clips the image if does not fit within the bounds specified by the button Size property value.

Example: `'icon.png'`

Example: `'C:\Documents\icon.png'`

`IconAlignment` — Location of button icon relative to Text
`'left'` (default) | `'center'` | `'top'` | `'bottom'` | `'right'`

Location of button icon relative to button Text (if any), specified as one of the following values:

Value	Result
`'left'`	 Left
`'center'`	 Center
`'top'`	 Top
`'bottom'`	 Bottom
`'right'`	 Right

Note: The preceding images reflect the icon location when HorizontalAlignment and VerticalAlignment are each set to their default value, `'center'`.

If you specify an empty string `('')` for the Text property, then the icon aligns as though it is text, according to the values set for HorizontalAlignment and VerticalAlignment. The IconAlignment property value has no effect in this case.

`Location` — Location of button relative to parent
[100,100] (default) | `[x,y]`

Location of button relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the button. The units of measurement are pixels.

`OuterLocation` — Location of button relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for buttons.

`OuterSize` — Size of button, including `Text` value
[100,20] (default) | `[width,height]`

Size of the button, returned as `[width,height]`. The units of measurement are pixels.

`Parent` — Parent container of button
app window object (default) | panel object | button group object

Parent container of the button, specified as an app window, panel, or button group object.

`Size` — Size of button
[100,20] (default) | `[width,height]`.

Size of the button, specified as `[width,height]`. The units of measurement are pixels.

`Text` — Characters that display on button
`'Button'` (default) | string | cell array of strings

Characters that display on button, specified as one of the following:

- string — Displayed as a single line string. For example:

```
uibutton('Text', 'Press');
```

- cell array of strings — Displayed as multiline text. Each row of the cell array is one line of text. Adjust button Size property value accordingly.

  ```
  uibutton('Text', {'Press','Here'}, 'Size',[40,40]);
  ```

Example: `'Restart Simulation'`

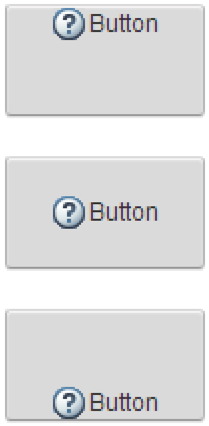
Example: `{'Restart', 'Simulation'}`

`VerticalAlignment` — Vertical alignment of text within button

`'center'` (default) | `'top'` | `'bottom'`

Vertical alignment of text within the button, specified as `'center'`, `'top'`, or `'bottom'`.

The following image shows the three vertical alignment options when the HorizontalAlignment is `'center'`.



Note: Vertical alignment values appear to have no effect when the button is the default height.

`Visible` — Button visibility

`true` (default) | `false`

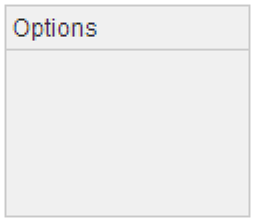
Button visibility, specified as `true` or `false`. The Visible property determines whether the button is displayed on the screen. If the Visible property of a button is set to `false`, the entire button is hidden, but you can still specify and access its properties.

Button Group Properties

Control button group appearance and behavior

Button groups are app components that manage the mutual exclusiveness of the radio buttons or toggle buttons that it contains. A button group can contain radio buttons or toggle buttons, but not both. Typically, the button group generates an action when the selected button changes. Properties control the appearance and behavior of a particular instance of a button group. To modify aspects of a button group, change property values. Use dot notation to refer to a particular object and property:

```
window = appwindow;
bg = matlab.ui.control.ButtonGroup('Parent',window);
title = bg.Title;
bg.Title = 'Options';
```



`BorderVisibility` — Whether the button group border is displayed

`true` (default) | `false`

Whether button group border is displayed, specified as `true` or `false`.

`Enabled` — Operational status of button group

`true` (default) | `false`

Operational status of button group, specified as `true` or `false.`

If you set this property to `true`, the button group appearance indicates that the app user can interact with child components (unless you disable individual components within the button group).

If you set this property to `false`, the button group appears dimmed, indicating that the app user cannot interact with child components, and the button group will not trigger a callback.

Note Changing the Enabled property of a button group does *not* change the values of the Enabled properties of its child components, even though disabling the button group causes the children to be disabled.

`FontAngle` — Character slant of button group title

`'normal'` (default) | `'italic'`

Character slant of the Button Group title, value specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display button group title

`'Helvetica'` (default) | string

Font in which to display the button group Title property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

Example: `'Arial'`

`FontSize` — Font size of button group title

12 (default) | positive number

Font size of button group title, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Location` — Location of button group relative to parent

`[100,100]` (default) | `[x,y]`

Location of button group relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the button group. The units of measurement are pixels.

`OuterSize` — Button Group size, including the title bar

[123,106] (default) | `[width,height]`

The button group size, including the title bar, returned as `[width,height]`. The units of measurement are pixels.

`Parent` — Parent container of button group

app window object (default) | panel object | button group object

Parent container of button group, specified as an app window, panel, or button group object.

`SelectedObject` — Currently selected radio or toggle button

first radio or toggle button in button group (default)

Currently selected radio button or toggle button, specified as a radio button or a toggle button object.

Get the value of this property to determine which radio or toggle button is currently selected within the button group.

Set the value of this property to change the currently selected radio or toggle button. When you change the selection using this property, MATLAB adjusts the Value property for the other radio or toggle buttons within the button group accordingly.

For example, suppose your button group contains three radio buttons and you set the SelectedObject property to `radiobutton3`. MATLAB sets the Value property for each child radio button as follows:

- radiobutton1.Value = `false;`
- radiobutton2.Value = `false;`
- radiobutton3.Value = `true;`

In other words, setting the SelectedObject property for a button group has the same effect as setting the Value property for a child button.

`SelectionChangeFcn` — Function that executes when an app user changes the button selection

`[ ]` (default) | function handle | cell array | string

Function that executes when an app user changes the button selection, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace

This callback function does not execute if a radio or toggle button Value property changes programmatically.

Example: @function

Example: {@function, x}

`Size` — Button Group size, excluding title bar

[123,85] (default) | [width,height]

The button group size, excluding the title bar, specified as `[width,height]`. The units of measurement are pixels.

`Title` — Button Group title

`'Button Group'` (default) | string | cell array of strings

The button group title, specified as a string.

`Visible` — Button Group visibility

`true` (default) | `false`

Button Group visibility, specified as `true` or `false`. The Visible property determines whether the button group is displayed on the screen. If the Visible property of a button group is set to `false`, the entire button group is hidden. However, you can still specify and access its properties.

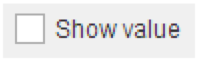
Note Changing the Visible property of a button group does *not* change the values of the Visible properties of its child components, even though hiding the button group causes the children to be hidden.

Check Box Properties

Control check box appearance and behavior

Check boxes are app components that indicate one of two states. Typically, check boxes generate an action when checked or cleared. Properties control the appearance and behavior of a particular instance of a check box. To modify aspects of a check box, change property values. Use dot notation to refer to a particular object and property:

```
checkbox = uicheckbox;
txt = checkbox.Text;
checkbox.Text = 'Show value';
```



`Enabled` — Operational state of check box

true (default) | false

Operational state of check box, specified as `true` or `false`.

If you set this property to `true`, the app user can select or clear it.

If you set this property to `false`, the check box appears dimmed, indicating that the app user cannot select or clear it and it will not trigger a callback.

`FontAngle` — Character slant of check box text

`'normal'` (default) | `'italic'`

Character slant of check box text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display check box text

`'Helvetica'` (default) | string

Font in which to display the check box Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of check box text

12 (default) | positive number

Font size of check box text, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Location` — Location of check box relative to parent container
[100,100] (default) | `[x,y]`

Location of check box relative to parent container, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container (panel or app window) to the lower-left corner of the Check Box. The units of measurement are pixels.

Example: `[100,150]`

`OuterLocation` — Location of check box relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for Check Boxes.

`OuterSize` — Size of check box, including `Text` value
[84,15] (default) | `[width,height]`

Identical to `Size` for Check Boxes.

`Parent` — Parent container of check box
app window object (default) | panel object | button group object

Parent container of check box, specified as an app window, panel, or button group object.

`Size` — Size of check box, including `Text` value
[84,15] (default) | `[width,height]`

Size of the check box, including the Text value, specified as `[width,height]`. The units of measurement are pixels.

`Text` — Check box label

`'Check Box'` (default) | string | cell array of strings

Check Box label, specified as one of the following:

- string — Displayed as a single line string. For example:

  ```
  uicheckbox('Text', 'Filter');
  ```

- cell array of strings — Displayed as multiline text. Each row of the cell array is one line of text. (You must increase the size from the default value to accommodate the additional line of text.)

  ```
  uicheckbox('Text',{'Notch','Filter'},'Size',[84,30]);
  ```

If you set the Text property to a cell array of strings, you must change the check box Size property value to accommodate the additional lines of text.

`Value` — State of check box

false (default) | true

State of check box specified as `true` or `false`. When the `Value` property is set to `true`, the check box is checked; when set to `false`, the check box is not checked.

`ValueChangedFcn` — Function that executes when app user selects or clears check box

[ ] (default) | function handle | cell array | string

Function that executes when app user selects or clears the check box, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

This callback function does not execute if the check box Value changes programmatically.

Example: @function

Example: {@function, x}

`Visible` — Check Box visibility

`true` (default) | `false`

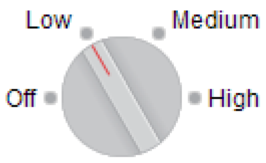
Check Box visibility, specified as `true` or `false`. The Visible property determines whether the check box is displayed on the screen. If the Visible property of a check box is set to `false`, the entire check box is hidden, but you can still specify and access its properties.

Discrete Knob Properties

Control discrete knob appearance and behavior

Discrete knobs are app components from which a user can select one option from a set. Typically, discrete knobs generate an action when a user changes the knob selection. Properties control the appearance and behavior of a particular instance of a discrete knob. To modify aspects of a discrete knob, change property values. Use dot notation to refer to a particular object and property:

```
knob = uiknob('discrete');
val = knob.Value;
knob.Value = 'Low';
```



`Enabled` — Operational state of discrete knob
`true` (default) | `false`

Operational state of discrete knob, specified as `true` or `false`.

If you set this property to `true`, the app user can turn the discrete knob.

If you set this property to `false`, the appearance of the discrete knob appears dimmed, indicating that the app user cannot turn the discrete knob and it will not trigger a callback.

`Location` — Location of knob, exclusive of state marks and labels, relative to parent
`[126,111]` (default) | `[x,y]`

Location of discrete knob, exclusive of state marks and labels, relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the discrete knob. The units of measurement are pixels.

`OuterLocation` — Location of knob, including state marks and labels, relative to parent
[101,110] (default) | `[x,y]`

Location of knob, including tick marks, relative to parent, specified as `[x,y]`. The units of measurement are pixels.

`OuterSize` — Size of knob, including state marks and labels

[127,86] (default) | `[height,width]`

Size of knob, including state marks and labels, returned as `[height,width]`. The units of measurement are pixels.

`Parent` — Parent container of discrete knob

app window object (default) | panel object | button group object

Parent container of discrete knob, specified as an app window, panel, or button group object.

`Size` — Size of knob, exclusive of state marks and labels

[60,60] (default) | `[width,height]`

Size of knob, exclusive of state marks and labels, specified as `[width,height]`. The units of measurement are pixels.

`Text` — Set of discrete knob labels

`{'Off','Low','Medium','High'}` (default) | 1-by-n cell array of strings

The set of discrete knob labels, specified as a 1-by-n cell array of strings. The cell array must contain at least two elements. The discrete knob displays as many options as there are elements in the Text property value. MATLAB labels the knob tick marks using the cell array values, applied in clockwise order on the knob.

Example: `{'Off','Slow','Fast'}`

Example: `{'1','2','3','4'}`

`TextData` — Data associated with each element of the `Text` property

`{1,2,3,4}` (default) | 1-by-n cell array of integers

Data associated with each element of the `Text` property, specified as a 1–by-n cell array of integers, where n is greater than 0. For example, if you set the Text property to temperature descriptions (`'boiling','lukewarm','freezing'`) you might set the TextData property to corresponding temperature values (`{212, 90, 32}` — in degrees in Fahrenheit).

If you change the number of elements in the `Text` property value, then the TextData property is updated accordingly, *unless* you previously explicitly set TextData. If you explicitly set the TextData property, then it is not updated automatically when the number of elements in the `Text` property changes.

If the TextData property value contains more elements than the Text property value, then MATLAB ignores the extra TextData property value elements.

If the TextData property value contains fewer elements than the Text property value, and an app user selects an option that has no corresponding TextData property element, then MATLAB sets the ValueData property to an empty cell array (`{}`) and MATLAB displays a warning at the command prompt.

`Value` — Value to which knob points

element of Text property array

The value to which the discrete knob points, specified as an element of the Text property array. The default is the first element of the Text property array.

If the Text property array is empty, MATLAB sets the Value property to an empty double array `[]`.

If the Text property array changes programmatically and the current Value setting is an element of the Text property array, then the Value remains unchanged.

If the Text property array changes programmatically and the Value setting is no longer an element of the Text property array, MATLAB sets Value to the first element of the Text property array.

If the ValueData property changes programmatically, MATLAB updates the Value property to the Text property array element corresponding to the new selection.

If the Value property changes programmatically to an element in ValueData that appears more than once in ValueData, then the item selected in the app corresponds to the first match.

`ValueChangedFcn` — Callback to execute when app user changes the knob control location

[ ] (default) | function handle | cell array | string

Callback to execute when the app user changes the discrete knob control setting, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments

- String that is a valid MATLAB expression, which is evaluated in the base workspace.

Example: @function

Example: {@function, x}

`ValueData` — Data associated with selected value

element of TextData

The data associated with the selected value, specified as an element of the TextData property value. MATLAB uses `isequal` to determine if the ValueData property value matches an element of the TextData property value. If TextData is empty, MATLAB sets the Value property to an empty double array `[]`.

If you programmatically set the ValueData property value, then the Value property setting and the selection in the app change accordingly.

If you programmatically set ValueData to an element in TextData that appears more than once in TextData, then the item selected in the app corresponds to the first match.

If you programmatically change TextData, the current selection in the app does not change. However, ValueData is updated to the new data of the selected option.

If the Value property setting changes, the ValueData property value is updated to the data associated with the new value.

`Visible` — Discrete Knob visibility

`true` (default) | `false`

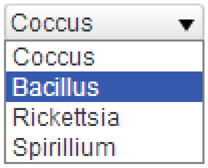
Discrete Knob visibility, specified as `true` or `false`. The Visible property determines whether the discrete knob is displayed on the screen. If the Visible property of a discrete knob is set to `false`, the entire discrete knob is hidden, but you can still specify and access its properties.

Drop Down Properties

Control drop down appearance and behavior

Drop downs are app components from which a user can select one option from a set. Until the user clicks the component, only the currently selected option is displayed. Typically, drop downs generate an action when a user changes the selection. Properties control the appearance and behavior of a particular instance of a drop down. To modify aspects of a drop down, change property values. Use dot notation to refer to a particular object and property:

```
dropdown = uidropdown;
txt = dropdown.Text;
dropdown.Text = {'Coccus', 'Bacillus',...
                    'Rickettsia', 'Spirillium'};
```



`Enabled` — Operational state of drop down

`true` (default) | `false`

Operational state of drop down, specified as `true` or `false`.

If you set this property to `true`, the appearance of the drop down indicates that the app user can open and make a selection from the drop down.

If you set this property to `false`, the appearance of the drop down appears dimmed, indicating that the app user cannot open or make a selection from the drop down and it will not trigger any callbacks.

`FontAngle` — Character slant of drop down text

`'normal'` (default) | `'italic'`

Character slant of drop down text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display drop down text

`'Helvetica'` (default) | string

Font in which to display the drop down Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of drop down text

12 (default) | positive number

Font size of drop down text, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Location` — Location of drop down relative to parent

[100,100] (default) | `[x,y]`

Location of drop down relative to parent, specified as `[x, y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the closed drop down. The units of measurement are pixels.

`OuterLocation` — Location of drop down relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for drop downs.

`OuterSize` — Size of drop down

[100, 20] (default) | `[width,height]`

Identical to `Size` for drop downs.

`Parent` — Parent container of drop down

app window object (default) | panel object | button group object

Parent container of drop down, specified as an app window, panel, or button group object.

`Size` — Size of drop down

[100,20] (default) | `[width,height]`.

Size of the drop down, specified as `[width,height]`. The units of measurement are pixels.

`Text` — Set of drop down options presented to app user

`{'Option 1','Option 2','Option 3','Option 4'}` (default) | 1-by-n cell array of strings

The set of drop down labels, specified as a 1-by-n cell array of strings. The drop down displays as many options as there are elements in the Text property value.

Example: `{'Red','Yellow','Blue'}`

Example: `{'1','2','3'}`

`TextData` — Set of drop down values

`{1, 2, 3, 4}` (default) | 1-by-n cell array

Data associated with each element of the `Text` property, specified as a 1-by-n cell array of integers, where n is greater than 0. For example, if you set the Text property to employee names, you might set the TextData property to corresponding employee id numbers.

If you change the number of elements in the `Text` property value, then the TextData property is updated accordingly, *unless* you previously explicitly set TextData. If you explicitly set the TextData property, then it is not updated automatically when the number of elements in the `Text` property changes.

If the TextData property value contains more elements than the Text property value, then MATLAB ignores the extra TextData property value elements.

If the TextData property value contains fewer elements than the Text property value, and an app user selects an option that has no corresponding TextData property element, then MATLAB sets the ValueData property to an empty cell array (`{}`). MATLAB displays a warning at the command prompt.

Example: `{'Red' 'Green' 'Blue'}`

Example: `{10 20 30 40}`

Example: `{true false}`

`Value` — Selected value
element of Text property cell array | string

Selected value, specified as an element of the Text property cell array.

The default is the first element of the Text property array.

If the Text property array is empty, MATLAB sets the Value property to an empty double array `[]`.

If the Text property array changes programmatically and the current Value setting is an element of the Text property array, then the Value remains unchanged.

If the Text property array changes programmatically and the Value setting is no longer an element of the Text property array, MATLAB sets Value to the first element of the Text property array.

If the ValueData property changes programmatically, MATLAB updates the Value property to the Text property array element corresponding to the new selection.

If the Value property changes programmatically to an element in ValueData that appears more than once in ValueData, then the item selected in the app corresponds to the first match.

`ValueChangedFcn` — Code to execute when app user changes drop down selection
`[]` (default) | function handle | cell array | string

Callback function to execute when the app user changes the drop down selection, specified as a function handle, a cell array containing a function handle and additional arguments, or a string.

This callback does not execute if the app user clicks the currently selected option, or if the Value property setting changes programmatically.

Example: @function

Example: {@function, x}

`ValueData` — Data associated with selected value
element of TextData

The data associated with the selected value, specified as an element of the TextData property value. MATLAB uses `isequal` to determine if the ValueData property value matches an element of the TextData property value. If TextData is empty, MATLAB sets the Value property to an empty double array `[]`.

If you programmatically set the ValueData property value, then the Value property setting and the selection in the app change accordingly.

If you programmatically set ValueData to an element in TextData that appears more than once in TextData, then the item selected in the app corresponds to the first match.

If you programmatically change TextData, the current selection in the app does not change. However, ValueData is updated to the new data of the selected option.

If the Value property setting changes, the ValueData property value is updated to the data associated with the new value.

`Visible` — Drop Down visibility

`true` (default) | `false`

Drop Down visibility, specified as `true` or `false`. The Visible property determines whether the drop down is displayed on the screen. If the Visible property of a drop down is set to `false`, the entire drop down is hidden, but you can still specify and access its properties.

Edit Field Properties

Control edit field appearance and behavior

Edit fields are app components into which app users can type text. Typically, edit fields generate an action when the user adds or changes text. Properties control the appearance and behavior of a particular instance of an edit field. To modify aspects of an edit field, change property values. Use dot notation to refer to a particular object and property:

```
txtfield = uieditfield;
txt = txtfield.Value;
txtfield.Value = 'Penicillin';
```

`Enabled` — Operational state of edit field

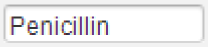
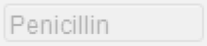
`true` (default) | `false`

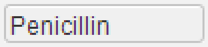
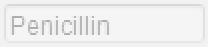
Operational state of edit field, specified as `true` or `false`.

If you set this property to `true`, the edit field accepts input.

If you set this property to `false`, the edit field appears dimmed, indicating that it does not accept input and the edit field will not trigger a callback.

As shown in the following table, the Editable property value also affects whether a edit field is operational.

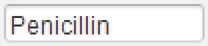
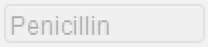
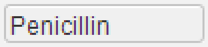
Enabled	Editable	Visual Result	Functional Result
`'true'`	`'true'`		App user can update the edit field. A callback associated with the edit field executes when text changes.
`'false'`	`'false'`		App user cannot update the edit field.

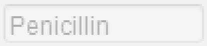
Enabled	Editable	Visual Result	Functional Result
`'true'`	`'false'`		App user cannot update the edit field. Use this combination of property values when you want the text to be easy to read, even though it is not editable.
`'false'`	`'true'`		App user cannot update the edit field.

`Editable` — Whether edit field is editable
`true` (default) | `false`

Whether edit field is editable, specified as `true` or `false`.

As shown in the following table, the Enabled property value also affects whether a edit field is editable.

Enabled	Editable	Visual Result	Functional Result
`'true'`	`'true'`		App user can update the edit field. A callback associated with the edit field executes when text changes.
`'false'`	`'false'`		App user cannot update the edit field.
`'true'`	`'false'`		App user cannot update the edit field. Use this combination of property values when you want the text to be easy to read, even though it is not editable.

Enabled	Editable	Visual Result	Functional Result
`'false'`	`'true'`	 Penicillin	App user cannot update the edit field.

`FontAngle` — Character slant of edit field text
`'normal'` (default) | `'italic'`

Character slant of edit field text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display edit field text
`'Helvetica'` (default) | string

Font in which to display the edit field Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of edit field text
12 (default) | positive number

Font size of edit field text, specified as a positive number. The units of measurement are points.

Example: 14

Data Types: `double`

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`HorizontalAlignment` — Horizontal alignment of text within edit field
`'left'` (default) | `'right'` | `'center'`

Alignment of text within the edit field, specified as `'left'`, `'right'`, or `'center'`. The alignment affects the display as the app user edits the edit field.

`Location` — Location of edit field relative to parent

[100,100] (default) | `[x,y]`

Location of edit field relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the edit field. The units of measurement are pixels.

`OuterLocation` — Location of edit field relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for Edit Fields.

`OuterSize` — Size of edit field

[100,20] (default) | `[width,height]`

Identical to `Size` for Edit Fields.

`Parent` — Parent container of edit field

app window object (default) | panel object | button group object

Parent container of the edit field, specified as an app window handle, panel, or button group object.

`Size` — Size of edit field

[100,20] (default) | `[width,height]`.

Size of the edit field, specified as `[width,height]`. The units of measurement are pixels.

`Value` — Text in text field

`''` (default) | string

Text in the edit field, specified as a string. MATLAB displays the string as a single line. If you want to allow multiple lines of text, create a Text Area instead.

Example: `'E. coli'`

`ValueChangedFcn` — Callback to execute after text entry or update

`[]` (default) | function handle | cell array | string

Callback function to execute after text entry or update, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

The callback executes after the app user changes text in the edit field, and then either presses **Enter**, or clicks outside the edit field.

Example: @function

Example: {@function, x}

`Visible` — Edit Field visibility

`true` (default) | `false`

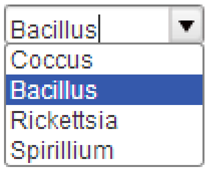
Edit Field visibility, specified as `true` or `false`. The Visible property determines whether the edit field is displayed on the screen. If the Visible property of a edit field is set to `false`, the entire edit field is hidden, but you can still specify and access its properties.

Editable Drop Down Properties

Control editable drop down appearance and behavior

Editable drop downs are app components from which a user can select one option from a set, or type in a string. Until the user clicks the down arrow, only the currently selected option is displayed. Typically, editable drop downs generate an action when a user changes the selection. Properties control the appearance and behavior of a particular instance of an editable drop down. To modify aspects of an editable drop down, change property values. Use dot notation to refer to a particular object and property:

```
edropdown = uidropdown('editable');
txt = edropdown.Text;
edropdown.Text = {'Coccus', 'Bacillus',...
                  'Rickettsia', 'Spirillium'};
```



`Enabled` — Whether editable drop down is operational

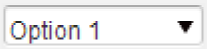
`true` (default) | `false`

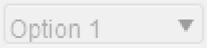
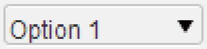
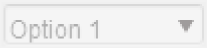
Whether editable drop down is operational, specified as `true` or `false`.

If you set this property to `true`, the appearance of the editable drop down is bright.

If you set this property to `false`, the appearance of the editable drop down appears dimmed.

As shown in the following table, the Editable property value also affects whether a editable drop down is operational.

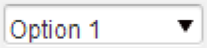
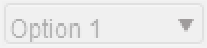
Enabled	Editable	Visual Result	Functional Result
`'true'`	`'true'`		App user can edit text or change the

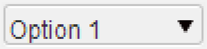
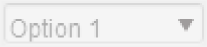
Enabled	Editable	Visual Result	Functional Result
			selection in the editable drop down. A callback associated with the editable drop down executes when an option changes.
`'false'`	`'false'`	 Option 1	App user cannot change the editable drop down selection or text.
`'true'`	`'false'`	 Option 1	App user cannot change the editable drop down selection or text.
`'false'`	`'true'`	 Option 1	App user cannot change the editable drop down selection or text.

`Editable` — Editable state of editable drop down
`true` (default) | `false`

Editable state of editable drop down, specified as `true` or `false`.

As shown in the following table, the Enabled property value also affects whether a editable drop down is editable.

Enabled	Editable	Visual Result	Functional Result
`'true'`	`'true'`	 Option 1	App user can edit text or change the selection in the editable drop down. A callback associated with the editable drop down executes when an option changes.
`'false'`	`'false'`	 Option 1	App user cannot change the editable drop down selection or text.

Enabled	Editable	Visual Result	Functional Result
`'true'`	`'false'`		App user cannot change the editable drop down selection or text.
`'false'`	`'true'`		App user cannot change the editable drop down selection or text.

`FontAngle` — Character slant of editable drop down text
`'normal'` (default) | `'italic'`

Character slant of editable drop down text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display editable drop down text
`'Helvetica'` (default) | string

Font in which to display the editable drop down Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of editable drop down text
12 (default) | positive number

Font size of editable drop down text, specified as a positive number. The units are pixels.

Example: 14

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Location` — Location of editable drop down relative to parent
[100,100] (default) | `[x,y]`

Location of editable drop down relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the editable drop down. The units of measurement are pixels.

`OuterLocation` — Location of editable drop down relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for editable drop downs.

`OuterSize` — Size of editable drop down

[100, 20] (default) | `[width,height]`

Identical to `Size` for editable drop downs.

`Parent` — Parent container of editable drop down

app window object (default) | panel object | button group object

Parent container of editable drop down, specified as an app window, panel, or button group object.

`Size` — Size of editable drop down

[100 20] (default) | `[width, height]`.

Size of the closed editable drop down, specified as `[width, height]`. The units of measurement are pixels.

`Text` — Set of editable drop down options presented to app user

`{'Option 1','Option 2','Option 3','Option 4'}` (default) | 1-by-n cell array of strings

The set of editable drop down labels, specified as a 1-by-n cell array of strings. The editable drop down displays as many options as there are elements in the Text property value.

Example: `{'Centigrade','Farenheit','kelvin'}`

Example: `{'1','2','3'}`

`TextData` — Set of Editable Drop Down values

`{1, 2, 3, 4}` (default) | 1-by-n cell array

Data associated with each element of the `Text` property, specified as a 1-by-n cell array of integers, where n is greater than 0. For example, if you set the Text property to employee names, you might set the TextData property to corresponding employee id numbers.

If you change the number of elements in the `Text` property value, then the TextData property is updated accordingly, *unless* you previously explicitly set TextData. If you explicitly set the TextData property, then it is not updated automatically when the number of elements in the `Text` property changes.

If the TextData property value contains more elements than the Text property value, then MATLAB ignores the extra TextData property value elements.

If the TextData property value contains fewer elements than the Text property value, and an app user selects an option that has no corresponding TextData property element, then MATLAB sets the ValueData property to an empty cell array (`{}`) and displays a warning at the command prompt.

Example: `{'Red' 'Green' 'Blue'}`

Example: `{'10' '20' '30' '40'}`

`Value` — Selected or typed value

`'Option 1'` (default) | element of Text property cell array | string

Selected or typed value, specified as an element of the Text property cell array or an app user-entered string.

If the Text property array is empty, MATLAB sets the Value property to the user-entered string or an empty double array `[]` when no string is entered.

If an app user types a string in the Editable Drop Down, the Value property setting becomes that string, the Text and TextData property value remains unchanged, and the ValueData property value is an empty vector.

If the Text property array changes programmatically and the current Value setting is an element of the Text property array, then the Value remains unchanged.

If the Text property array changes programmatically and the Value setting is no longer an element of the Text property array, then MATLAB sets Value to the first element of the Text property array.

If the ValueData property changes programmatically, MATLAB updates the Value property to the Text property array element corresponding to the new selection.

If the Value property changes programmatically to an element in ValueData that appears more than once in ValueData, then the item selected in the app corresponds to the first match.

`ValueChangedFcn` — Code to execute when app user changes the editable drop down option
[ ] (default) | function handle | cell array | string

Callback function to execute when the app user changes theeditable drop down option (by selecting a new option or entering text), specified as a function handle, a cell array containing a function handle and additional arguments, or a string.

This callback does not execute if the app user clicks the currently selected option, or if the Value property setting changes programmatically.

Example: @function

Example: {@function, x}

`ValueData` — Data associated with specified value
element of TextData | [ ]

The data associated with specified value, specified as one of the following:

- An element of TextData

  MATLAB uses `isequal` to determine if the ValueData property value matches an element of the TextData property value.
- An empty double array (`[ ]`), if the TextData property is empty

If you programmatically change the ValueData property value, then the selection in the app changes accordingly.

If you programmatically set ValueData to an element in TextData that appears more than once in TextData, then the item selected in the app corresponds to the first match.

If you programmatically change TextData, the current selection in the app does not change. However, ValueData is updated to the new data of the selected option.

If Value property setting changes, the ValueData property value is updated to the data associated with the new value.

`Visible` — Editable Drop Down visibility
`true` (default) | `false`

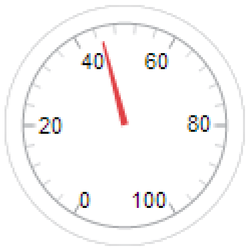
Editable Drop Down visibility, specified as `true` or `false`. The Visible property determines whether the editable drop down is displayed on the screen. If the Visible property of a editable drop down is set to `false`, the entire editable drop down is hidden, but you can still specify and access its properties.

Gauge Properties

Control gauge appearance and behavior

Gauges are app components that represent a measurement instrument. Properties control the appearance and behavior of a particular instance of a gauge. To modify aspects of a gauge, change property values. Use dot notation to refer to a particular object and property:

```
gauge = uigauge;
val = gauge.Value;
gauge.Value = 45;
```



`Enabled` — Operational state of gauge

`true` (default) | `false`

Operational state of gauge, specified as `true` or `false`.

If you set this property to `true`, the appearance of the gauge indicates that thegauge is operational.

If you set this property to `false`, the appearance of the gauge appears dimmed, indicating that the gauge is not operational. However, app code can change the gauge property values.

`Limits` — Minimum and maximum gauge scale values

[0,100] (default) | array

Minimum and maximum gauge scale values, specified as a 2-element numeric array. The first value in the array must be lower than the second value.

`Location` — Location of gauge relative to parent container

[100,100] (default) | `[x,y]`

Location of gauge relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of an imaginary box surrounding the gauge. The units of measurement are pixels.

`MajorTicks` — Major tick mark values

[0, 20,40,60,80,100] (default) | array of numeric values

Major tick mark values, specified as an array of numeric values. Any array values that fall outside Limits property value are not displayed on the gauge. MATLAB removes duplicate array values.

Setting the MajorTicks property value sets the MajorTicksMode property value to `'manual'`.

`MajorTickLabels` — Major tick mark labels

`{'0','20','40','60','80','100'}` (default) | cell array of strings

Major tick mark labels, specified as a cell array of strings. MATLAB uses each element as the label for the corresponding major tick mark.

Setting the MajorTickLabels property value changes the MajorTickLabelsMode property value to `'manual'`.

To create a blank label to a major tick, specify an empty string for the corresponding MajorTickLabels property value element.

If you specify more MajorTickLabels property value elements than MajorTicks property value elements, MATLAB ignores the extra labels.

If you specify fewer MajorTickLabels property value elements than MajorTicks property value elements, MATLAB leaves the extra ticks unlabeled.

`MajorTickLabelsMode` — Major tick label creation mode

`'auto'` (default) | `'manual'`

Major tick label creation mode, specified as one of the following:

- `'auto'` — MATLAB creates a cell array of strings to populate the major tick labels. MATLAB transforms each numeric major tick element to a string using `num2str`.

- `'manual'`- You specify the major tick labels by specifying the MajorTickLabels property value.

`MajorTicksMode` — Major tick creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB populates the MajorTicks property value by creating several major tick marks in the range specified by the Limits property (inclusive).
- `'manual'`- You specify the major ticks by specifying the MajorTicks property value.

`MinorTicks` — Minor tick values

[0:4:100] (default) | numeric array

Minor tick values, specified as a numeric array. Any MinorTicks elements that fall outside Limits are not displayed on the gauge. If a minor tick falls on the same value as a major tick, only the major tick is displayed. MATLAB removes duplicate MinorTicks element values. Setting the MinorTicks property value sets the MinorTicksMode property value to `'manual'`.

To exclude minor ticks from the gauge, specify an empty array.

`MinorTicksMode` — Minor tick creation mode

`'auto'` (default) | `'manual'`

Minor tick creation mode, specified as one of the following:

- `'auto'` — Minor ticks display at positions around the gauge determined by calculating the difference between the largest major tick interval between any two major consecutive major ticks and dividing it by five. For example, if major ticks are uniformly distributed, the resulting gauge has four minor ticks between every major tick.

  Minor ticks depend only on the visible major ticks (that is, Major Ticks within scale limits).

  MATLAB does not generate minor ticks for Major Ticks that are beyond scale limits.

  When the Limits property value changes, minor ticks are updated to populate the full scale range (the MinorTicks property is updated accordingly).

- `'manual'`- You specify the MinorTicks property numeric array. The MinorTicks property value does not change size or content on its own.

`OuterLocation` — Location of gauge relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for Gauges.

`OuterSize` — Size of gauge
[120,120] (default) | `[width,height]`

Identical to the `Size` property for Gauges.

`Parent` — Parent container of gauge
app window object (default) | panel object | button group object

Parent container of gauge, specified as an app window, panel, or button group object.

`ScaleColorLimits` — Range of values specifying start and end of colored scale regions
[ ] (default) | array

Range of values specifying start and end of colored scale regions, specified as a n-by-2 array of numeric values. For every row in the array, the first element must be less than the second element.

When applying colors to the gauge's scale, MATLAB applies the colors starting at the first row in the ScaleColors property value. Therefore, if two rows in ScaleColorLimits property value overlap, then the color applied later takes precedence.

A gauge does not display any portion of a ScaleColorLimits property value element that falls outside of Limits property value.

If ScaleColors and ScaleColorLimits property values are different sizes, then the gauge shows only the colors that have matching limits. For example, if the ScaleColors property value has 3 rows, but the ScaleColorLimits property value has only 2 rows, the gauge displays the first two color/ limit pairs only.

Example: [10,20; 20,30]

`ScaleColors` — Colors that appear on gauge scale
[ ] (default) | RGB triplet | predefined color name | 1-by-n cell array

Colors that appear on gauge scale, specified as one of the following:

- An n-by-3 array of RGB values, where each element is a valid RGB value between 0 and 1.
- A 1-by-n cell array, where each element is a color name or an RGB triplet. The cell array can contain a mix of RGB triplets and color names.

  MATLAB stores and returns all color names and RGB triplets as an n-by-3 array.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Each row of the ScaleColors array represents the color for the *i* the colored scale region.

If you set ScaleColors property without explicitly setting ScaleColorLimits property, then MATLAB determines the ScaleColorLimits property value by creating n equally spaced, nonoverlapping limits that span the gauge’s entire scale. For example, if the Limits property value is `[0,75]`, and the ScaleColors property value is `{'green','yellow','red'}`, then MATLAB sets ScaleColorLimits to `[0,25; 25,50; 50,75]`

Similarly, if you change Limits, without explicitly setting ScaleColorLimits, then MATLAB sets the ScaleColorLimits property value as described in the preceding paragraph.

Example: `{'blue','green'}`

Example: `{[0,.5,.2], 'green'}`

Example: `{[0,.5,.2], [.2,.6,0]}`

`ScaleDirection` — Direction of scale

`'clockwise'` (default) | `'counterclockwise'`

Direction of scale, specified as one of the following:

- `'clockwise'` — The scale appears such that the scale tick values increase in a clockwise manner.
- `'counterclockwise'` — The scale appears such that the scale tick values increase in a counterclockwise manner.

`Size` — Gauge size

[120,120] (default) | `[width,height]`

Gauge size, specified as [width,height]. The units of measurement are pixels.

`Value` — gauge needle location

0 (default) | numeric

The gauge needle location, specified as any numeric value. If the value is less than the minimum Limits property value, then the needle points to a location immediately before the beginning of the scale. If the value is more than the maximum Limits property value, then the needle points to a location immediately after the end of the scale.

Changing the Limits property value has no effect on the Value property setting.

Example: `60`

`Visible` — Gauge visibility

`true` (default) | `false`

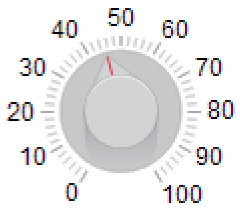
Gauge visibility, specified as `true` or `false`. The Visible property determines whether the gauge is displayed on the screen. If the Visible property of a gauge is set to `false`, the entire gauge is hidden, but you can still specify and access its properties.

Knob Properties

Control knob appearance and behavior

Knobs are app components representing instrument control knobs that app users can adjust to control a value. Properties control the appearance and behavior of a particular instance of a knob. To modify aspects of a knob, change property values. Use dot notation to refer to a particular object and property:

```
knob = uiknob;
knob.Value = 45;
val = knob.Value;
```



`Enabled` — Visual appearance and behavior

`true` (default) | `false`

Visual appearance and behavior of the knob, specified as `true` or `false`.

If you set this property to `true`, the knob appearance is bright, indicating that the app user can move the it.

If you set this property to `false`, the knob appears dimmed, indicating that the app user cannot move it and it will not trigger a callback.

`Limits` — Minimum and maximum knob values

`[0,100]` (default) | array

Minimum and maximum knob values, specified as a 2-element numeric array.

If you change Limits such that the knob Value is less than the new lower limit, MATLAB sets the Value to the new lower limit. For example, suppose Limits is `[0,100]` and the knob Value is 20. If the Limits changes to `[50,100]`, then MATLAB sets the knob Value to 50.

Likewise, if you change Limits such that the knob Value is greater than the new upper limit, MATLAB sets the Value to the new upper limit.

`Location` — Location of knob, exclusive of tick marks, relative to parent
[126,111] (default) | `[x,y]`

Location of knob, exclusive of tick marks, relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the knob. The units of measurement are pixels.

`MajorTicks` — Major tick mark values
[0,10,20,30,40,50,60,70,80,90,100] (default) | array of numeric values

Major tick mark values, specified as an array of numeric values. Each value must be equal to or greater than 0. Any array values that fall outside Limits are not displayed on the knob. MATLAB removes duplicate array values.

Setting the MajorTicks property value sets the MajorTicksMode property value to `'manual'`.

`MajorTickLabels` — Labels on each major tick mark
`{'0','10','20','30','40','50','60','70','80','90','100'}` (default) | cell array of strings

Labels on each major tick mark, specified as a cell array of strings. MATLAB uses each element as the label for the corresponding major tick mark.

Setting MajorTickLabels changes the MajorTickLabelsMode value to `'manual'`.

To create a blank label to a major tick, specify an empty string for that tick's corresponding MajorTickLabels element.

If you specify more MajorTickLabels elements than MajorTicks elements, MATLAB ignores the extra labels.

If you specify fewer MajorTickLabels elements than MajorTicks elements, MATLAB leaves the extra ticks unlabeled.

Example: `{'high','medium', 'low'}`

`MajorTickLabelsMode` — Major tick labels creation mode
`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB creates a cell array of strings to populate MajorTickLabels. MATLAB transforms each numeric major tick element to a string using `num2str`.
- `'manual'`- You specify the MajorTickLabels cell array of strings.

`MajorTicksMode` — Major tick creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB populates MajorTicks by creating several major tick marks in the range specified by Limits (inclusive).
- `'manual'`— You specify the MajorTicks value array.

`MinorTicks` — Minor tick mark values

[0:2:100] (default) | numeric array

Minor tick mark values, specified as a numeric array. Any MinorTicks elements that fall outside Limits are not displayed on the gauge. If a minor tick falls on the same value as a major tick, only the major tick is displayed. MATLAB removes duplicate MinorTicks element values. Setting the MinorTicks property value sets the MinorTicksMode property value to `'manual'`.

To exclude minor ticks from the gauge, specify an empty array.

`MinorTicksMode` — Minor tick creation mode

`'auto'` (default) | `'manual'`

Minor tick creation mode, specified as one of the following:

- `'auto'` — MATLAB displays minor ticks at positions around the Knob determined by calculating the difference between the largest major tick interval between any two major consecutive major ticks and dividing it by five. For example, if major ticks are uniformly distributed, the resulting knob has four minor ticks between every major tick.

  Minor ticks depend only on the visible major ticks (that is, Major Ticks within scale limits).

  MATLAB does not generate minor ticks for Major Ticks that are beyond scale limits.

When the Limits value changes, minor ticks are updated to populate the full scale range (the MinorTicks property is updated accordingly).

- `'manual'`- You specify the MinorTicks numeric array. The MinorTicks value does not change size or content on its own.

`OuterLocation` — Location of knob, including tick marks, relative to parent

[100,104] (default) | `[x,y]`

Location of knob, including tick marks, relative to parent, specified as `[x,y]`. The units are pixels.

`OuterSize` — Size of knob, including tick marks

[112,100] (default) | `[height,width]`

Size of knob, including tick marks, returned as `[height,width]`.

`Parent` — Parent container of knob

app window object (default) | panel object | button group object

Parent container of knob, specified as an app window handle, panel, or button group object.

`Size` — Size of knob, exclusive of tick marks

[60,60] (default) | `[width,height]`

Size of knob, exclusive of tick marks, specified as `[width,height]`.

`Value` — Knob value

0 (default) | numeric

The knob value, specified as a numeric. The numeric value must be within the range specified for Limits.

`ValueChangedFcn` — Callback function to execute when app user changes knob value

`[ ]` (default) | function handle | cell array | string

Callback function to execute once when an app user changes the knob value (by dragging and releasing the knob), specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments

- String that is a valid MATLAB expression, which is evaluated in the base workspace.

Example: @function

Example: {@function, x}

`ValueChangingFcn` — Callback function to execute as the app user changes knob value
[ ] (default) | function handle | cell array | string

The callback function to execute as the app user changes the knob value, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

When an app user drags the knob, MATLAB generates event data and stores it in the event `Value` property, which you can query.

For example, to display the event value in the MATLAB Command Window, specify this code in the `ValueChangingFcn` callback:

```
disp(event.Value)
```

The callback executes as follows:

- If an app user clicks the knob value, the callback executes once. For example, if the knob is on 1.0, and the user single clicks 1.1, the callback executes and the event value is updated once, with a value of 1.1.
- If an app user clicks and drags the knob to a new position, the callback executes repeatedly. For example, if the knob value is 1.0, and the user clicks, holds, and drags to value 1.1, the callback executes and the event `Value` is updated to 1.1. If the user continues the drag to 1.2, the callback executes again and the event `Value` is updated to 1.2. When the user releases the mouse, the callback does not execute.
- If the knob Value property changes programmatically, then the callback does not execute.

Example: @function

Example: {@function, x}

`Visible` — Knob visibility
`true` (default) | `false`

Knob visibility, specified as `true` or `false`. The Visible property determines whether the knob is displayed on the screen. If the Visible property of a knob is set to `false`, the entire knob is hidden, but you can still specify and access its properties.

Label Properties

Control label appearance

Labels are app components that use text to identify other components or specify the purpose of those other components. To modify aspects of a label, change property values. Use dot notation to refer to a particular object and property:

```
label = uilabel;
label.Text = 'Type:';
txt = label.Text;
```

Type:

`Enabled` — Visual appearance of label

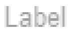
`true` (default) | `false`

Visual appearance of label, specified as one of the following:

- `true` — Label is displayed normally

Label

- `false` — Label is displayed dimmed


Label

`FontAngle` — Character slant of label text

`'normal'` (default) | `'italic'`

Character slant of label text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user’s system.

`FontName` — Font in which to display label `Text`

`'Helvetica'` (default) | string

Font in which to display the label Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of label text

12 (default) | positive number

Font size of label text, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`HorizontalAlignment` — Horizontal alignment of label text

`'left'` (default) | `'right'` | `'center'`

Alignment of label text, specified as:

- `'right'` — Text aligns on the right side of the area specified by the label Size property.
- `'left'` — Text aligns on the left side of the area specified by the label Size property.
- `'center'`— Text centers horizontally in the area specified as the label Size property.

Aligning label text is useful when the text spans multiple lines.

`Location` — Location of label relative to parent

[100,100] (default) | `[x,y]`

Location of label relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the label. The units of measurement are pixels.

`OuterLocation` — Location of label relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for Labels.

`OuterSize` — Size of label

[31,15] (default) | `[width,height]`

Identical to `Size` for Labels.

`Parent` — Label container

app window object (default) | panel object | button group object

The label container, specified as an app window, panel, or button group object.

`Size` — Size of display area for label

[31,15] (default) | `[width,20]`.

Size of the display area for the label, specified as `[width,height]`. If the `Size` property value is too small to display the Text value, MATLAB crops the text in the app. The units of measurement are pixels.

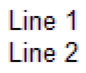
`Text` — Label text

`'Label'` (default) | string | cell array of strings

The label text, specified as a string or a cell array of strings. Use a cell array of strings to specify a multiline label.

MATLAB can properly render formatted text, such as this:

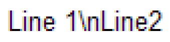
```
text = sprintf('%s\n%s', 'Line 1', 'Line 2')
label = uilabel('Text', text, 'Size',[100,100])
```



```
Line 1
Line 2
```

However, MATLAB does not automatically interpret and render text such as this:

```
label = uilabel('Text', 'Line 1\nLine2', 'Size',[100,150])
```



```
Line 1\nLine2
```

Example: `'Threshold'`

Example: `{'Threshold' 'Value'}`

`VerticalAlignment` — Vertical alignment of text

`'top'` (default) | `'bottom'` | `'center'`

Vertical alignment of label text, specified as one of the following:

- `'top'` — Text aligns on the top of the area specified by the Label Size property.
- `'bottom'` — Text aligns on the bottom of the area specified by the Label Size property.
- `'center'` —Text is centered vertically in the center of the area specified by the label Size property.

Aligning label text is useful when the text spans multiple lines.

`Visible` — Label visibility

`true` (default) | `false`

Label visibility, specified as `true` or `false`. The Visible property determines whether the label is displayed on the screen. If the Visible property of a label is set to `false`, the entire label is hidden, but you can still specify and access its properties.

Lamp Properties

Control lamp appearance

Lamps are app components that indicate state using color. Lamp properties control the appearance of a particular instance of a lamp. To modify aspects of a lamp, change property values. Use dot notation to refer to a particular object and property:

```
lamp = uilamp;
color = lamp.Color;
lamp.Color = 'red';
```

`Color` — lamp color

[0,1,0] (green) (default) | RGB triplet | predefined color name

Lamp color, specified as one of the following:

- RGB triplet
- Predefined color name

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Regardless of how you specify the colors, MATLAB stores and returns them as RGB triplets. For example:

```
lamp = uilamp('Color','blue');
x = lamp.Color
```

```
ans =
     0     0     1
```

Example: `[0,1,0]`

Example: `'g'`

Example: `'green'`

`Enabled` — Operational state of lamp
`true` (default) | `false`

Operational state of lamp, specified as `true` or `false`.

If you set this property to `true`, the lamp appears bright, indicating that the lamp is operational.

If you set this property to `false`, the Lamp appears dimmed.

`Location` — Location of lamp relative to parent
[100,100] (default) | `[x,y]`

Location of lamp relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the lamp. The units of measurement are pixels.

`OuterLocation` — Location of lamp relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for Lamps.

`OuterSize` — Size of lamp
[20,20] (default) | `[width,height]`

Identical to `Size` for Lamps.

`Parent` — Parent container of lamp
app window object (default) | panel object | button group object

Parent container of lamp, specified as an app window, panel, or button group object.

`Size` — Size of lamp

[20,20] (default) | `[width,height]`.

Size of the lamp, specified as `[width, height]`. The units of measurement are pixels.

`Visible` — Lamp visibility

`true` (default) | `false`

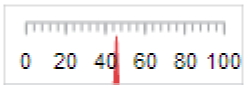
Lamp visibility, specified as `true` or `false`. The Visible property determines whether the lamp is displayed on the screen. If the Visible property of a lamp is set to `false`, the entire lamp is hidden, but you can still specify and access its properties.

Linear Gauge Properties

Control linear gauge appearance and behavior

Linear degree gauges are app components that represent a measurement instrument. Properties control the appearance and behavior of a particular instance of a linear gauge. To modify aspects of a linear gauge, change property values. Use dot notation to refer to a particular object and property:

```
gauge = uigauge('linear');
val = gauge.Value;
gauge.Value = 45;
```



`Enabled` — Operational state of linear gauge

`true` (default) | `false`

Operational state of linear gauge, specified as `true` or `false`.

If you set this property to `true`, the appearance of the linear gauge indicates that thelinear gauge is operational.

If you set this property to `false`, the appearance of the linear gauge appears dimmed, indicating that the linear gauge is not operational. However, app code can change the linear gauge property values.

`Limits` — Minimum and maximum linear gauge scale values

[0,100] (default) | array

Minimum and maximum linear gaugescale values, specified as a 2-element numeric array. The first value in the array must be lower than the second value.

`Location` — Location of linear gauge relative to parent

[100,100] (default) | `[x,y]`

Location of linear gauge relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the linear gauge. The units of measurement are pixels.

`MajorTicks` — Major tick mark values

[0,20,40,60,80,100] (default) | array of numeric values

Major tick mark values, specified as an array of numeric values. Each value must be equal to or greater than 0. Any array values that fall outside Limits property value are not displayed on the linear gauge. MATLAB removes duplicate array values.

Setting the MajorTicks property value sets the MajorTicksMode property value to `'manual'`.

`MajorTickLabels` — Major tick mark labels

`{'0','20','40','60','80','100'}` (default) | cell array of strings

Major tick mark labels, specified as a cell array of strings. MATLAB uses each element as the label for the corresponding major tick mark.

Setting the MajorTickLabels property value changes the MajorTickLabelsMode property value to `'manual'`.

To create a blank label to a major tick, specify an empty string for the corresponding MajorTickLabels property value element.

If you specify more MajorTickLabels property value elements than MajorTicks property value elements, MATLAB ignores the extra labels.

If you specify fewer MajorTickLabels property value elements than MajorTicks property value elements, MATLAB leaves the extra ticks unlabeled.

`MajorTickLabelsMode` — Major tick label creation mode

`'auto'` (default) | `'manual'`

Major tick label creation mode, specified as one of the following:

- `'auto'` — MATLAB creates a cell array of strings to populate the major tick labels. MATLAB transforms each numeric major tick element to a string using `num2str`.
- `'manual'` — You specify the major tick labels by specifying the MajorTickLabels property value.

`MajorTickLabels` — Major tick creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB populates the MajorTicks property value by creating 6 equally spaced major tick marks in the range specified by the Limits property (inclusive).
- `'manual'` — You specify the major ticks by specifying the MajorTicks property value.

`MinorTicks` — Minor tick values
[0:4:100] (default) | numeric array

Minor tick values, specified as a numeric array. Any MinorTicks elements that fall outside Limits are not displayed on the gauge. If a minor tick falls on the same value as a major tick, only the major tick is displayed. MATLAB removes duplicate MinorTicks element values. Setting the MinorTicks property value sets the MinorTicksMode property value to `'manual'`.

To exclude minor ticks from the gauge, specify an empty array.

`MinorTicksMode` — Minor tick creation mode
`'auto'` (default) | `'manual'`

Minor tick creation mode, specified as one of the following:

- `'auto'` — Minor ticks display at positions across the gauge determined by calculating the difference between the largest major tick interval between any two major consecutive major ticks and dividing it by five. For example, if major ticks are uniformly distributed, the resulting gauge has four minor ticks between every major tick.

  Minor ticks depend only on the visible major ticks (that is, Major Ticks within scale limits).

  MATLAB does not generate minor ticks for Major Ticks that are beyond scale limits.

  When the Limits property value changes, minor ticks are updated to populate the full scale range (the MinorTicks property is updated accordingly).
- `'manual'` — You specify the MinorTicks property numeric array. The MinorTicks property value does not change size or content on its own.

`Orientation` — Layout position of linear gauge
`'horizontal'` (default) | `'vertical'`

Layout position of linear gauge, specified as `'horizontal'` or `'vertical'`.

`OuterLocation` — Location of linear gauge relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for Linear Gauges.

`OuterSize` — Size of linear gauge

[120,40] (default) | `[width,height]`

Identical to the `Size` property for Linear Gauges.

`Parent` — Parent container of linear gauge

app window object (default) | panel object | button group object

Parent container of linear gauge, specified as an app window, panel, or button group object.

`ScaleColorLimits` — Range of values specifying start and end of colored scale regions

[ ] (default) | array

Range of values specifying start and end of colored scale regions, specified as a n-by-2 array of numeric values. For every row in the array, the first element must be less than the second element.

When applying colors to the linear gauge's scale, MATLAB applies the colors starting at the first row in the ScaleColors property value. Therefore, if two rows in ScaleColorLimits property value overlap, then the color applied later takes precedence.

A gauge does not display any portion of a ScaleColorLimits property value element that falls outside of Limits property value.

If ScaleColors and ScaleColorLimits property values are different sizes, then the gauge shows only the colors that have matching limits. For example, if the ScaleColors property value has 3 rows, but the ScaleColorLimits property value has only 2 rows, the gauge displays the first two color/ limit pairs only.

Example: [10,20; 20,30]

`ScaleColors` — Colors that appear on gauge scale

[ ] (default) | RGB triplet | predefined color name | 1-by-n cell array

Colors that appear on linear gauge scale, specified as one of the following:

- An n-by-3 array of RGB values, where each element is a valid RGB value between 0 and 1.

- A 1-by-n cell array, where each element is a color name or an RGB triplet. The cell array can contain a mix of RGB triplets and color names.

  MATLAB stores and returns all color names and RGB triplets as an n-by-3 array.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Each row of the ScaleColors array represents the color for the *i*th colored scale region.

If you set ScaleColors property without explicitly setting ScaleColorLimits property, then MATLAB determines the ScaleColorLimits property value by creating n equally spaced, nonoverlapping limits that span the linear gauge's entire scale. For example, if the Limits property value is `[0,75]`, and the ScaleColors property value is `{'green', 'yellow', 'red'}`, then MATLAB sets ScaleColorLimits to `[0,25; 25,50; 50,75]`

Similarly, if you change Limits, without explicitly setting ScaleColorLimits, then MATLAB sets the ScaleColorLimits property value as described in the preceding paragraph.

Example: `{'blue', 'green'}`

Example: `{[0,.5,.2], 'green'}`

Example: `{[0,.5,.2], [.2,.6,0]}`

`Size` — Size of linear gauge

[120,40] (default) | `[width,height]`

Size of the linear gauge, specified as `[width,height]`. The units of measurement are pixels.

`Value` — gauge needle location

0 (default) | numeric

The linear gauge needle location, specified as any numeric value. If the value is less than the minimum Limits property value, then the needle points to a location immediately before the beginning of the scale. If the value is more than the maximum Limits property value, then the needle points to a location immediately after the end of the scale.

Changing the Limits property value has no effect on the Value property setting.

Example: `60`

`Visible` — Linear Gauge visibility

`true` (default) | `false`

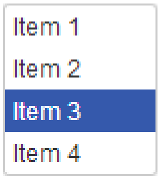
Linear Gauge visibility, specified as `true` or `false`. The Visible property determines whether the linear gauge is displayed on the screen. If the Visible property of a linear gauge is set to `false`, the entire linear gauge is hidden, but you can still specify and access its properties.

List Box Properties

Control list box appearance and behavior

List boxes are app components that display a list of items from which an app user can select one or more items. Typically, list boxes generate an action when one or more items are selected. Properties control the appearance and behavior of a particular instance of a list box. To modify aspects of a list box, change property values. Use dot notation to refer to a particular object and property:

```
listbox = uilistbox;
selection = listbox.Value;
listbox.Value = 'Item 3';
```



`Enabled` — Operational state of list box

`true` (default) | `false`

Operational state of list box, specified as `true` or `false`.

If you set this property to `true`, an app user can make a selection from the list box.

If you set this property to `false`, the list box appears dimmed, indicating that an app user cannot make a selection and the list box will not trigger a callback.

`Location` — Location of list box relative to parent

[100,100] (default) | `[x,y]`

Location of the list box relative to the parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the list box. The units of measurement are pixels.

`Multiselect` — If app user can select multiple list box elements

`false` (default) | `true`

If app user can select multiple list box elements, specified as `true` or `false`.

`OuterLocation` — Location of list box relative to parent
[100,100] (default) | `[x,y]`

Identical to `Location` for List Boxes.

`OuterSize` — Size of list box
[77,86] (default) | `[width,height]`.

Identical to `Size` for List Boxes.

`Parent` — Parent container of List Box
app window object (default) | panel object | button group object

Parent container of List Box, specified as an app window, panel, or button group object.

`Size` — Size of list box
[77,86] (default) | `[width,height]`.

Size of the list box, specified as `[width,height]`. The units of measurement are pixels.

`Text` — List Box options presented to app user
`{'Item 1','Item 2', 'Item 3', 'Item 4'}` (default) | 1-by-n cell array of strings

The list box options presented to the app user, specified as a 1-by-n cell array of strings. Duplicate elements are allowed. MATLAB displays the Text property cell array elements from top to bottom in the list box.

Each string corresponds to a TextData property cell array element. For example, if you set the Text property to display employee names, you might set the TextData property to corresponding employee id numbers.

If you specify fewer elements for the Text property cell array than for the TextData property cell array, MATLAB ignores the extra elements specified for the TextData property cell array.

If you specify more elements for the Text property cell array than for the TextData property cell array and an app user selects an option that has no corresponding TextData property cell array element, then MATLAB sets the `ValueData` property as follows, depending on the value of the `Multiselect` property:

- If the `Multiselect` property is set to `true`, then the `ValueData` property is set to an empty double array (`[]`).

- If the `Multiselect` property is set to `false`, then the `ValueData` property is set to an empty cell array (`{}`).

an empty double array if the `Multiselect` property value is false,

`TextData` — Set of List Box values

`{1, 2, 3, 4}` (default) | 1-by-n cell array | empty cell array (`{}`)

The set of list box values, specified as a 1-by-n cell array or an empty cell array. Duplicate elements are allowed.

If the number of elements in the Text property changes, MATLAB automatically updates the TextData cell array to {1, 2, 3, … N}, where N is the number of elements in the Text property cell array. *However*, if you explicitly set the TextData property, it ceases to update automatically as the Text property cell array changes.

Example: `{'Red', 'Green', 'Blue'}`

Example: `{10, 20, 30, 40}`

Example: `{true, false}`

`Value` — List Box selection

`'Item 1'` (default) | element or elements of Text property cell array

List Box selection, specified as an element or elements (if the Multiselect property is set to `true`) of the Text property cell array.

The default is the first element of the Text property cell array. The Value property setting must be an element of the Text property cell array, as determined by `isequal`.

If the Text property cell array is empty, MATLAB sets the Value property depending on the value of the Multiselect property, as follows:

- If the Multiselect property is set to `false`, MATLAB sets the Value property to an empty double array `[]`.
- If the Multiselect property is set to `true`, MATLAB sets the Value property to an empty cell array (`{}`).

Setting the Value property updates the selection displaying in the running app. Therefore, if Value is set to an element that appears more than once in the Text property cell array, then the selected item corresponds to the first match.

If the Multiselect property is set to `true`, then the selected items correspond to the elements that match. The matching elements can be repeated in the Value property value up to the number of times they appear in the Text property cell array.

If code changes the Text property cell array elements, it has one of the following effects:

- If the currently selected option in the running app is an element of the updated Text property cell array, then the Value property setting remain unchanged.
- If the currently selected value in a running app is *not* an element of the updated Text cell array, then MATLAB sets the Value property to the first element of the Text property cell array.

If the ValueData property changes, MATLAB updates the Value property to the Text cell array element corresponding to the new ValueData property value.

`ValueChangedFcn` — Callback function that executes when app user changes list box selections

[ ] (default) | function handle | cell array | string

Callback function that executes when app user changes list box selections, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

Example: @function

Example: {@function, x}

`ValueData` — Data associated with the list box selection

1 (default) | element of TextData

Data associated with the list box selection, specified as an element or elements (if the Multiselect property is set to `true`) of the TextData property cell array.

If the TextData property cell array is empty, valid values for the ValueData property depends on the MultiSelect property value, as follows:

- If the Multiselect property is set to `false`, the ValueData property must be an empty double array `[]`.

- If the Multiselect property is set to `true`, the ValueData property must be an empty cell array (`{}`).

`Visible` — List Box visibility

`true` (default) | `false`

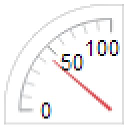
List Box visibility, specified as `true` or `false`. The Visible property determines whether the list box is displayed on the screen. If the Visible property of a list box is set to `false`, the entire list box is hidden, but you can still specify and access its properties.

Ninety Degree Gauge Properties

Control ninety degree gauge appearance and behavior

Ninety degree gauges are app components that represent a measurement instrument. Properties control the appearance and behavior of a particular instance of a ninety degree gauge. To modify aspects of a ninety degree gauge, change property values. Use dot notation to refer to a particular object and property:

```
gauge = uigauge('ninetydegree');
val = gauge.Value;
gauge.Value = 45;
```



`Enabled` — Operational state of ninety degree gauge

`true` (default) | `false`

Operational state of ninety degree gauge, specified as `true` or `false`.

If you set this property to `true`, the appearance of the ninety degree gauge indicates that theninety degree gauge is operational.

If you set this property to `false`, the appearance of the ninety degree gauge appears dimmed, indicating that the ninety degree gauge is not operational. However, app code can change the ninety degree gauge property values.

`Limits` — Minimum and maximum ninety degree gauge scale values

[0,100] (default) | array

Minimum and maximum ninety degree gauge scale values, specified as a 2-element numeric array. The first value in the array must be lower than the second value.

`Location` — Location of ninety degree gauge relative to parent

[100,100] (default) | `[x,y]`

Location of ninety degree gauge relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the ninety degree gauge. The units of measurement are pixels.

If the orientation of the gauge is such that a rounded edge is closest to the lower left corner of the parent container, then`x` and `y` specify the coordinates to the southwest corner of an imaginary box surrounding the gauge.

`MajorTicks` — Major tick mark values

[0,50,100] (default) | array of numeric values

Major tick mark values, specified as an array of numeric values. Each value must be equal to or greater than 0. Any array values that fall outside Limits property value are not displayed on the ninety degree gauge. MATLAB removes duplicate array values.

Setting the MajorTicks property value sets the MajorTicksMode property value to `'manual'`.

`MajorTickLabels` — Major tick mark labels

`{'0', ''50','100']}` (default) | cell array of strings

Major tick mark labels, specified as a cell array of strings. MATLAB uses each element as the label for the corresponding major tick mark.

Setting the MajorTickLabels property value changes the MajorTickLabelsMode property value to `'manual'`.

To create a blank label to a major tick, specify an empty string for the corresponding MajorTickLabels property value element.

If you specify more MajorTickLabels property value elements than MajorTicks property value elements, MATLAB ignores the extra labels.

If you specify fewer MajorTickLabels property value elements than MajorTicks property value elements, MATLAB leaves the extra ticks unlabeled.

`MajorTickLabelsMode` — Major tick label creation mode

`'auto'` (default) | `'manual'`

Major tick label creation mode, specified as one of the following:

- `'auto'` — MATLAB creates a cell array of strings to populate the major tick labels. MATLAB transforms each numeric major tick element to a string using `num2str`.
- `'manual'` — You specify the major tick labels by specifying the MajorTickLabels property value.

`MajorTicksMode` — Major tick creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB populates the MajorTicks property value by creating several major tick marks in the range specified by the Limits property (inclusive).
- `'manual'` — You specify the major ticks by specifying the MajorTicks property value.

`MinorTicks` — Minor tick values

[0:10:100] (default) | numeric array

Minor tick values, specified as a numeric array. Any MinorTicks elements that fall outside Limits are not displayed on the gauge. If a minor tick falls on the same value as a major tick, only the major tick is displayed. MATLAB removes duplicate MinorTicks element values. Setting the MinorTicks property value sets the MinorTicksMode property value to `'manual'`.

To exclude minor ticks from the gauge, specify an empty array.

`MinorTicksMode` — Minor tick creation mode

`'auto'` (default) | `'manual'`

Minor tick creation mode, specified as one of the following:

- `'auto'` — Minor ticks display at positions around the gauge determined by calculating the difference between the largest major tick interval between any two major consecutive major ticks and dividing it by five. For example, if major ticks are uniformly distributed, the resulting gauge has four minor ticks between every major tick.

  Minor ticks depend only on the visible major ticks (that is, Major Ticks within scale limits).

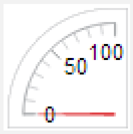
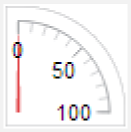
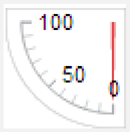
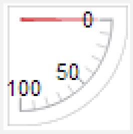
  MATLAB does not generate minor ticks for Major Ticks that are beyond scale limits.

  When the Limits property value changes, minor ticks are updated to populate the full scale range (the MinorTicks property is updated accordingly).
- `'manual'` — You specify the MinorTicks property numeric array. The MinorTicks property value does not change size or content on its own.

`Orientation` — Layout position of ninety degree gauge

`'northwest'` (default) | `'northeast'` | `'southwest'` | `'southeast'`

Layout position of ninety degree gauge, specified as one of the following positions:

`'northwest'`	
`'northeast'`	
`'southwest'`	
`'southeast'`	

`OuterLocation` — Location of ninety degree gauge relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for Ninety Degree Gauges.

`OuterSize` — Size of ninety degree gauge
[60,60] (default) | `[width,height]`

Identical to the `Size` property for Ninety Degree Gauges.

`Parent` — Parent container of ninety degree gauge
app window object (default) | panel object | button group object

Parent container of ninety degree gauge, specified as an app window, panel, or button group object.

`ScaleColorLimits` — Range of values specifying start and end of colored scale regions
[ ] (default) | array

Range of values specifying start and end of colored scale regions, specified as a n-by-2 array of numeric values. For every row in the array, the first element must be less than the second element.

When applying colors to the ninety degree gauge's scale, MATLAB applies the colors starting at the first row in the ScaleColors property value. Therefore, if two rows in ScaleColorLimits property value overlap, then the color applied later takes precedence.

A gauge does not display any portion of a ScaleColorLimits property value element that falls outside of Limits property value.

If ScaleColors and ScaleColorLimits property values are different sizes, then the gauge shows only the colors that have matching limits. For example, if the ScaleColors property value has 3 rows, but the ScaleColorLimits property value has only 2 rows, the gauge displays the first two color/ limit pairs only.

Example: [10,20; 20,30]

`ScaleColors` — Colors that appear on gauge scale

[ ] (default) | RGB triplet | predefined color name | 1-by-n cell array

Colors that appear on ninety degree gauge scale, specified as one of the following:

- An n-by-3 array of RGB values, where each element is a valid RGB value between 0 and 1
- A 1-by-n cell array, where each element is a color name or an RGB triplet. The cell array can contain a mix of RGB triplets and color names.

  MATLAB stores and returns all color names and RGB triplets as an n-by-3 array.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`

Long Name	Short Name	RGB Triplet
'black	'k'	[0 0 0]

Each row of the ScaleColors array represents the color for the *i*th colored scale region.

If you set ScaleColors property without explicitly setting ScaleColorLimits property, then MATLAB determines the ScaleColorLimits property value by creating n equally spaced, nonoverlapping limits that span the ninety degree gauge's entire scale. For example, if the Limits property value is `[0,75]`, and the ScaleColors property value is `{'green', 'yellow', 'red'}`, then MATLAB sets ScaleColorLimits to `[0,25; 25,50; 50,75]`

Similarly, if you change Limits, without explicitly setting ScaleColorLimits, then MATLAB sets the ScaleColorLimits property value as described in the preceding paragraph.

Example: `{'blue', 'green'}`

Example: `{[0,.5,.2], 'green'}`

Example: `{[0,.5,.2], [.2,.6,0]}`

`ScaleDirection` — Direction of scale

`'clockwise'` (default) | `'counterclockwise'`

Direction of scale, specified as one of the following:

- `'clockwise'` — The scale appears such that the scale tick values increase in a clockwise manner.
- `'counterclockwise'` — The scale appears such that the scale tick values increase in a counterclockwise manner.

`Size` — Size of ninety degree gauge

[60,60] (default) | `[width,height]`.

Size of the ninety degree gauge, specified as `[width,height]`. The width and height of the gauge is determined by the length of the two straight edges of the gauge, regardless of orientation. The units of measurement are pixels.

`Value` — gauge needle location

0 (default) | numeric

The ninety degree gauge needle location, specified as any numeric value. If the value is less than the minimum Limits property value, then the needle points to a location immediately before the beginning of the scale. If the value is more than the maximum Limits property value, then the needle points to a location immediately after the end of the scale.

Changing the Limits property value has no effect on the Value property setting.

Example: `60`

`Visible` — Ninety Degree Gauge visibility

`true` (default) | `false`

Ninety Degree Gauge visibility, specified as `true` or `false`. The Visible property determines whether the ninety degree gauge is displayed on the screen. If the Visible property of a ninety degree gauge is set to `false`, the entire ninety degree gauge is hidden, but you can still specify and access its properties.

Numeric Edit Field Properties

Control numeric edit field appearance and behavior

Numeric edit fields are app components that enable app users to enter numeric values. Typically, numeric edit fields generate an action when an app user changes the value. Properties control the appearance and behavior of a particular instance of a numeric edit field. To modify aspects of a numeric edit field, change property values. Use dot notation to refer to a particular object and property:

```
numberfield = uieditfield('numeric');
val = numberfield.Value;
numberfield.Value = 20;
```

`Enabled` — Operational state of numeric edit field

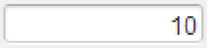
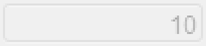
`true` (default) | `false`

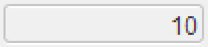
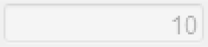
Operational state of numeric edit field, specified as `true` or `false`.

If you set this property to `true`, the numeric edit field accepts input.

If you set this property to `false`, the appearance of the numeric edit field appears dimmed indicating that it does not accept input and the numeric edit field will not trigger a callback.

As shown in the following table, the Editable property value also affects whether a numeric edit field is operational.

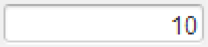
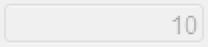
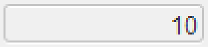
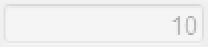
Enabled	Editable	Visual Result	Functional Result
`'true'`	`'true'`		App user can update the edit field. A callback associated with the edit field will execute when text changes.
`'false'`	`'false'`		App user cannot update the edit field.

Enabled	Editable	Visual Result	Functional Result
`'true'`	`'false'`	 10	App user cannot update the edit field.
`'false'`	`'true'`	 10	App user cannot update the edit field.

`Editable` — Whether numeric edit field is editable
`true` (default) | `false`

Whether numeric edit field is editable, specified as `true` or `false`.

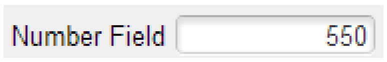
As shown in the following table, the Enabled property value also affects whether a numeric edit field is editable.

Enabled	Editable	Visual Result	Functional Result
`'true'`	`'true'`	 10	App user can update the edit field. A callback associated with the edit field will execute when text changes.
`'false'`	`'false'`	 10	App user cannot update the edit field.
`'true'`	`'false'`	 10	App user cannot update the edit field.
`'false'`	`'true'`	 10	App user cannot update the edit field.

`HorizontalAlignment` — Horizontal alignment of numbers within numeric edit field
`'right'` (default) | `'left'`

Horizontal alignment of numbers within the numeric edit field, specified as:

- `'right'` — Numbers align on the right side of the numeric edit field.



- `'left'` — Numbers align on the left side of the numeric edit field.

`Limits` — Minimum and maximum numeric edit field values
[0,100] (default) | array

Minimum and maximum values, specified as a 2-element numeric array. The first value must be lower than the second value. Set array elements to `-Inf` or `Inf` to specify no minimum or no maximum, respectively.

Example: [-Inf,200]

Example: [-100,Inf]

Example: [-100,200]

`Location` — Location of numeric edit field relative to parent
[100,100] (default) | `[x,y]`

Location of numeric edit field relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the numeric edit field. The units of measurement are pixels.

`LowerLimitInclusivity` — Whether `Value` property setting can be equal to `Limit` property lower value
`false` (default) | `true`

Whether Value property setting can be equal to the `Limits` property lower value, specified as one of the following:

- `true` — Value property setting must be equal to or greater than the lower limit
- `false` — Value property setting must be greater than the lower limit

`OuterLocation` — Location of numeric edit field relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for Numeric Edit Fields.

`OuterSize` — Size of numeric edit field, including `Text` value
[100,20] (default) | `[width,height]`

Identical to `Size` for Numeric Edit Fields.

`Parent` — Parent container of numeric edit field

app window object (default) | panel object | button group object

Parent container of numeric edit field, specified as an app window, panel, or button group object.

`RoundFractionalValues` — Whether MATLAB changes fractional value entered by app user to whole number

`false` (default) | `true`

Whether MATLAB changes a fractional value entered by the app user to a whole number, specified as one of the following:

- `true` — MATLAB rounds the value if it results in a valid value, and then executes the ValueChangedFcn callback. If the resulting value is outside the lower or upper limits, then MATLAB reverts the value to the preceding value, and does not execute the ValueChangedFcn callback.
- `false` — MATLAB does not change a user-specified fractional value to a whole number.

If the RoundFractionalValues property value changes from `false` to `true` in a running app, then MATLAB applies these rules:

- If rounding the existing value yields an integer that lies inside the limit range specified by the Limits property, then MATLAB rounds up the existing value.
- If rounding the existing value yields an integer that is less than the lower limit, then MATLAB rounds up the existing value.
- If rounding the existing value yields an integer that is greater than the upper limit, then MATLAB rounds down the existing value.
- If the limits are configured such that here is no valid integer in the range, then MATLAB sets RoundFractionalValues back to false and displays an error message.

`Size` — Size of numeric edit field

[100,20] (default) | `[width,height]`

Size of the numeric edit field, specified as `[width,height]`. The units of measurement are pixels.

`UpperLimitInclusivity` — Whether `Value` property setting can be equal to `Limit` property upper value

`false` (default) | `true`

Whether Value can be equal to the Limits property upper value, specified as one of the following:

- `true` — Value property setting must be equal to or less than the upper limit
- `false` — Value property setting must be less than the upper limit

For example, if you want the numeric input to be between 0 and 1, excluding 0 and 1, set the Limit property value to `[0,1]`, the UpperLimitInclusivity property to `false` and the LowerLimitInclusivity property to `false`.

`Value` — Value in numeric edit field

0 (default) | double-precision number

Value in the numeric edit field, specified as a double-precision number.

When an app user types or changes a value in the numeric edit field it is a string. When the user presses the **Enter** key or changes focus, MATLAB converts the user-entered string to a double-precision number.

MATLAB rejects the value if:

- It cannot convert the string to a scalar number.
- The value is NaN, blank, or a complex number.
- The value is a mathematical expression, such as `1 + 2`.
- The value is less than the Limit property lower limit or greater than the upper limit.

When MATLAB rejects a user-entered value, a tooltip appears describing the value requirements. The edit field text immediately reverts to the previous numeric edit field value. In this case, no ValueChangedFcn fires.

Example: 10

`ValueChangedFcn` — Code to execute when app user changes numeric edit field value

function handle (default) | function handle | cell array | string

Code to execute when app user changes the numeric edit field value, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

This function executes whenever an app user changes the value in the numeric edit field and changes focus or presses the **Enter** key. This function does not execute if the numeric edit field value changes programmatically.

Example: @function-name

Example: {@function-name, argument}

`ValueDisplayFormat` — Format of numeral in edit field

`'%11.4g'` (default) | format string

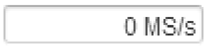
The format of the numeral in the numeric edit field, specified as a format string.

MATLAB uses `num2str(NumericDisplayFormat,Value)` to convert the edit field `Value` property to a string and displays it.

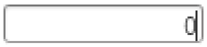
You can mix text with format strings. For example:

```
numfield = uieditfield('numeric','ValueDisplayFormat','%.0f MS/s');
```

The resulting numeric edit field looks like this image:



When the user clicks in the numeric edit field, the field shows the value without the string.



For a complete list of supported format strings, see `num2str` in the MATLAB documentation.

Example: '%.0f MS/s'

`Visible` — Numeric Edit Field visibility

`true` (default) | `false`

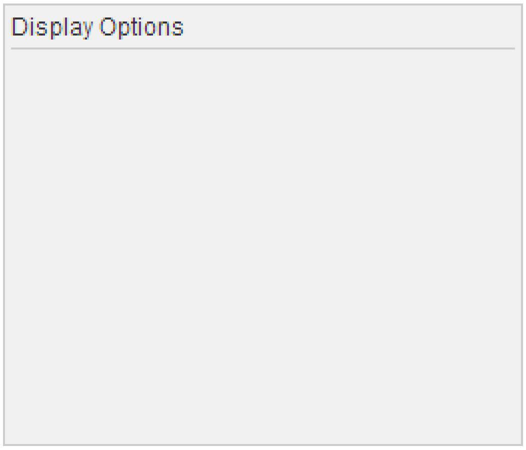
Numeric Edit Field visibility, specified as `true` or `false`. The Visible property determines whether the numeric edit field is displayed on the screen. If the Visible property of a numeric edit field is set to `false`, the entire numeric edit field is hidden, but you can still specify and access its properties.

Panel Properties

Control panel appearance

Panels are app components that contain other components. Typically, you use them to organize components in your app. Properties control the appearance and behavior of a particular instance of a panel. To modify aspects of a panel, change property values. Use dot notation to refer to a particular object and property:

```
window = appwindow;
panel = matlab.ui.control.Panel('Parent', window);
title = panel.Title;
panel.Title = 'Display Options';
```



`BorderVisibility` — Whether panel border is displayed
`true` (default) | `false`

Whether panel border is displayed, specified as `true` or `false`.

`Children` — Children of panel
empty `GraphicsPlaceholder` array (default) | 1-D array of component objects

Children of panel, returned as an empty `GraphicsPlaceholder` or a 1-D array of component objects. The children of a panel can be any component, including another panel.

You cannot add or remove children components using the `Children` property of the panel. Use this property to view the list of children or to reorder the children. The order of the children in this array reflects the front-to-back order (stacking order) of the components on the screen.

To add a child to this list, set the `Parent` property of the child component to this panel.

`Enabled` — Operational status of panel
`true` (default) | `false`

Visual appearance of panel, specified as

- `true` — Panel and its child components (unless you disable a child component within it) appear operational.
- `false` — Panel and its child components appear dimmed and you cannot manipulate any components within it.

Note Changing the Enabled property of a panel does *not* change the values of the Enabled properties of its child components, even though disabling the panel causes the children to be disabled.

`FontAngle` — Character slant of panel title
`'normal'` (default) | `'italic'`

Character slant of the Panel title, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display panel title
`'Helvetica'` (default) | string

Font in which to display the panel Title property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

Example: `'Arial'`

`FontSize` — Font size of panel title
12 (default) | positive number

Font size of panel title, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Location` — Location of panel relative to parent

[100,100] (default) | `[x,y]`

Location of panel relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the panel. The units of measurement are pixels.

`OuterLocation` — Location of panel relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for Panels.

`OuterSize` — Panel size, including the title bar

[260,221] (default) | `[width,height]`

The panel size, including the title bar, returned as `[width,height]`. The units of measurement are pixels.

`Parent` — Parent container of panel

app window object (default) | panel object | button group object

Parent container of panel, specified as an app window, panel, or button group object.

`Size` — Panel size, excluding the title bar

[260,200] (default) | `[width,height]`

The panel size, excluding the title bar, specified as `[width,height]`. The units of measurement are pixels.

`Title` — Panel title

`'Panel'` (default) | string

The panel title, specified as a string.

`Visible` — Panel visibility

`'on'` (default) | `'off'`

Panel handle visibility, specified as `'on'` or `'off'`. The Visible property determines whether the panel is displayed on the screen. If the Visible property of a panel is set to `'off'`, the entire panel is hidden, but you can still specify and access its properties.

Note Changing the Visible property of a panel does *not* change the values of the Visible properties of its child components, even though hiding the panel causes the components to be hidden.

Radio Button Properties

Control radio button appearance

Radio buttons are app components that appear in a button group, as a set of mutually exclusive options. Typically, the button group to which the radio button belongs generates an action when the selection changes. Properties control the appearance and behavior of a particular instance of a radio button. To modify aspects of a radio button, change property values. Use dot notation to refer to a particular object and property:

```
radiobutton = uiradiobutton;
txt = radiobutton.Text;
radiobutton.Text = 'One';
```

Note: To specify how you want your app to respond when a user changes a radio button selection, code the ValueChangedFcn callback for the Button Group to which the radio buttons are parented.

`Enabled` — Operational state of radio button

`true` (default) | `false`

Operational state of radio button, specified as `true` or `false`.

If you set this property to `true`, the appearance of the radio button indicates that an app user can select it.

If you set this property to `false`, the appearance of the radio button appears dimmed, indicating that an app user cannot select it.

`FontAngle` — Character slant of radio button text

`'normal'` (default) | `'italic'`

Character slant of the radio button text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user’s system.

`FontName` — Font in which to display radio button text

`'Helvetica'` (default) | string

Font in which to display the radio button Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

Example: `'Arial'`

`FontSize` — Font size of radio button text

12 (default) | positive number

Font size of radio button text, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Location` — Location of radio button relative to button group

[10,10] (default) | `[x,y]`

Location of the radio button relative to the parent button group container, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the button group container to the lower-left corner of the Radio Button. The units of measurement are pixels.

`OuterLocation` — Location of radio button relative to parent container

[10,10] (default) | `[x,y]`

Identical to `Location` for Radio Buttons.

`OuterSize` — Size of radio button, including text

[123,106] (default) | [width, height]

Identical to `Size` for Radio Buttons.

`Parent` — Parent button group container of radio button
button group object

Parent button group container of radio button, specified as a button group object. For details, see Button Group.

`Size` — Radio Button size, including text
[123,85] (default) | `[width,height]`

The radio button size, including Text property value, specified as `[width, height]`. The units of measurement are pixels.

`Text` — Radio Button text
`'Radio Button'` (default) | string

The radio button text, specified as a string.

Example: `'Steady state'`

`Value` — Selection state of radio button
`true` | `false`

Selection state of radio button, specified as `true` or `false`.

Within a given radio button group, one and only one radio button can be selected at a time.

When the Value property for a radio button changes to `true`, the Value property for the previously selected radio button parented to the same button group changes to `false`. In addition, the SelectedObject property value of the button group is updated to reflect the selected radio button.

If you programmatically change the Value property for a radio button to `false`, MATLAB sets the first radio button in the button group to `true`. If the first radio button is the one for which you set the Value property for a radio button to `false`, then MATLAB sets the second radio button in the button group to `true`.

`Visible` — Radio Button visibility
`true` (default) | `false`

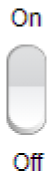
Radio Button visibility, specified as `true` or `false`. The Visible property determines whether the radio button is displayed on the screen. If the Visible property of a radio button is set to `false`, the entire radio button is hidden, but you can still specify and access its properties.

Rocker Switch Properties

Control rocker switch appearance and behavior

Rocker switches are app components that indicate a logical state. Typically, rocker switches generate an action when the state changes. Properties control the appearance and behavior of a particular instance of a rocker switch. To modify aspects of a rocker switch, change property values. Use dot notation to refer to a particular object and property:

```
rswitch = uiswitch('rocker');
val = rswitch.Value;
rswitch.Value = true;
```



`Enabled` — Operational state of rocker switch
`true` (default) | `false`

Operational state of rocker switch, specified as `true` or `false`.

If you set this property to `true`, the app user can slide the rocker switch.

If you set this property to `false`, the rocker switch appears dimmed, indicating that an app user cannot slide the rocker switch and it will not trigger a callback.

`Location` — Location of rocker switch, exclusive of state labels, relative to parent
`[100,121]` (default) | `[x,y]`

Location of rocker switch, exclusive of state labels, relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the rocker switch. The units of measurement are pixels.

`Orientation` — Direction in which rocker switch is displayed
`'vertical'` (default) | `'horizontal'`

Direction in which rocker switch is displayed, specified as `'horizontal'` or `'vertical'`.

`OuterLocation` — Location of rocker switch, inclusive of state labels, relative to parent
`[100,100]` (default) | `[x,y]`

Location of rocker switch, inclusive of state labels, relative to parent, returned as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the rocker switch, including state labels. The units of measurement are pixels.

`OuterSize` — Size of switch, inclusive of state labels
`[20,87]` (default) | `[width,height]`

Size of rocker switch, inclusive of state labels, returned as `[width,height]`.

`Parent` — Parent container of rocker switch
app window object (default) | panel object | button group object

Parent container of rocker switch, specified as an app window, panel, or button group object.

`Size` — Size of rocker switch, exclusive of state labels
`[20,45]` | `[width,height]`

Size of rocker switch, exclusive of state labels, specified as `[width,height]`. The units of measurement are pixels.

`Text` — Rocker Switch state labels
`{'Off', 'On'}` (default) | n-by-2 cell array strings

Rocker Switch state labels, specified as a n-by-2 cell array of strings.

When the rocker switch Orientation property is set to `'vertical'`, the first element of the array appears on the bottom of the rocker switch. When the Orientation property is set to `'horizontal'`, the first element of the array appears to the left of the rocker switch.

`Value` — State of rocker switch
`false` (default) | `true`

State of rocker switch, specified as one of the following values:

- `false`

The rocker switch points down when the Orientation property is set to `'vertical'`. It points left when the Orientation property is set to `'horizontal'`.

- `true`

  The rocker switch points up when the Orientation property is set to `'vertical'`. It points right when the Orientation property is set to `'horizontal'`.

`ValueChangedFcn` — Code to execute when app user flips rocker switch

[ ] (default) | function handle | cell array | string

Code to execute when app user flips the rocker switch control, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

The app user can change the rocker switch state by clicking and releasing the mouse button anywhere on the rocker switch (including the state labels), or by clicking on the rocker switch, dragging, and then releasing the mouse button while still on the rocker switch.

Example: @function

Example: {@function, x}

`Visible` — Rocker Switch visibility

`true` (default) | `false`

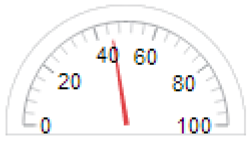
Rocker Switch visibility, specified as `true` or `false`. The Visible property determines whether the rocker switch is displayed on the screen. If the Visible property of a rocker switch is set to `false`, the entire rocker switch is hidden, but you can still specify and access its properties.

Semicircular Gauge Properties

Control semicircular gauge appearance

Semicircular gauges are app components that represent a measurement instrument. Properties control the appearance and behavior of a particular instance of a semicircular gauge. To modify aspects of a semicircular gauge, change property values. Use dot notation to refer to a particular object and property:

```
gauge = uigauge('semicircular');
val = gauge.Value;
gauge.Value = 45;
```



`Enabled` — Operational state of semicircular gauge
`true` (default) | `false`

Operational state of semicircular gauge, specified as `true` or `false`.

If you set this property to `true`, the appearance of the semicircular gauge indicates that thesemicircular gauge is operational.

If you set this property to `false`, the appearance of the semicircular gauge appears dimmed, indicating that the semicircular gauge is not operational. However, app code can change the semicircular gauge property values.

`Limits` — Minimum and maximum semicircular gauge scale values
[0,100] (default) | array

Minimum and maximum semicircular gaugescale values, specified as a 2-element numeric array. The first value in the array must be lower than the second value.

`Location` — Location of semicircular gauge relative to parent container
[100,100] (default) | `[x,y]`

Location of semicircular gauge relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the semicircular gauge. The units of measurement are pixels.

If the orientation of the gauge is such that a rounded edge is closest to the lower left corner of the parent container, then`x` and `y` specify the coordinates to the southwest corner of an imaginary box surrounding the gauge.

`MajorTicks` — Major tick mark values

[0,10,20,30,40,50,60,70,80,90,100] (default) | array of numeric values

Major tick mark values, specified as an array of numeric values. Each value must be equal to or greater than 0. Any array values that fall outside Limits property value are not displayed on the semicircular gauge. MATLAB removes duplicate array values.

Setting the MajorTicks property value sets the MajorTicksMode property value to `'manual'`.

`MajorTickLabels` — Major tick mark labels

`{'0','10','20','30','40','50','60','70','80','90','100'}` (default) | cell array of strings

Major tick mark labels, specified as a cell array of strings. MATLAB uses each element as the label for the corresponding major tick mark.

Setting the MajorTickLabels property value changes the MajorTickLabelsMode property value to `'manual'`.

To create a blank label to a major tick, specify an empty string for the corresponding MajorTickLabels property value element.

If you specify more MajorTickLabels property value elements than MajorTicks property value elements, MATLAB ignores the extra labels.

If you specify fewer MajorTickLabels property value elements than MajorTicks property value elements, MATLAB leaves the extra ticks unlabeled.

`MajorTicksMode` — Major tick creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB populates the MajorTicks property value by creating several major tick marks in the range specified by the Limits property (inclusive).
- `'manual'` — You specify the major ticks by specifying the MajorTicks property value.

`MajorTickLabelsMode` — Major tick label creation mode

`'auto'` (default) | `'manual'`

Major tick label creation mode, specified as one of the following:

- `'auto'` — MATLAB creates a cell array of strings to populate the major tick labels. MATLAB transforms each numeric major tick element to a string using `num2str`.
- `'manual'` — You specify the major tick labels by specifying the MajorTickLabels property value.

`MinorTicks` — Minor tick values

[0:4:100] (default) | numeric array

Minor tick values, specified as a numeric array. Any MinorTicks elements that fall outside Limits are not displayed on the gauge. If a minor tick falls on the same value as a major tick, only the major tick is displayed. MATLAB removes duplicate MinorTicks element values. Setting the MinorTicks property value sets the MinorTicksMode property value to `'manual'`.

To exclude minor ticks from the gauge, specify an empty array.

`MinorTicksMode` — Minor tick creation mode

`'auto'` (default) | `'manual'`

Minor tick creation mode, specified as one of the following:

- `'auto'` — Minor ticks display at positions around the gauge determined by calculating the difference between the largest major tick interval between any two major consecutive major ticks and dividing it by five. For example, if major ticks are uniformly distributed, the resulting gauge has four minor ticks between every major tick.

  Minor ticks depend only on the visible major ticks (that is, Major Ticks within scale limits).

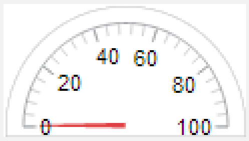
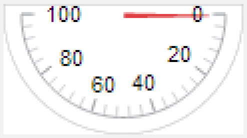
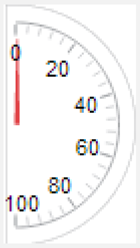
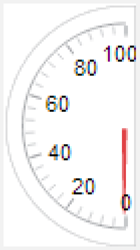
  MATLAB does not generate minor ticks for Major Ticks that are beyond scale limits.

  When the Limits property value changes, minor ticks are updated to populate the full scale range (the MinorTicks property is updated accordingly).
- `'manual'` — You specify the MinorTicks property numeric array. The MinorTicks property value does not change size or content on its own.

`Orientation` — Layout position of semicircular gauge

`'north'` (default) | `'south'` | `'east'` | `'west'`

Layout position of semicircular gauge, specified as one of the following:

`'north'`	
`'south'`	
`'east'`	
`'west'`	

`OuterLocation` — Location of semicircular gauge relative to parent container

[100,100] (default) | `[x,y]`

Identical to `Location` for Semicircular Gauges.

`OuterSize` — Size of semicircular gauge

[120,65] (default) | `[width,height]`

Identical to the `Size` property for Semicircular Gauges.

`Parent` — Parent container of semicircular gauge

app window object (default) | panel object | button group object

Parent container of semicircular gauge, specified as an app window, panel, or button group object.

`ScaleColorLimits` — Range of values specifying start and end of colored scale regions

[ ] (default) | array

Range of values specifying start and end of colored scale regions, specified as a n-by-2 array of numeric values. For every row in the array, the first element must be less than the second element.

When applying colors to the semicircular gauge's scale, MATLAB applies the colors starting at the first row in the ScaleColors property value. Therefore, if two rows in ScaleColorLimits property value overlap, then the color applied later takes precedence.

A gauge does not display any portion of a ScaleColorLimits property value element that falls outside of Limits property value.

If ScaleColors and ScaleColorLimits property values are different sizes, then the gauge shows only the colors that have matching limits. For example, if the ScaleColors property value has 3 rows, but the ScaleColorLimits property value has only 2 rows, the gauge displays the first two color/ limit pairs only.

Example: [10,20; 20,30]

`ScaleColors` — Colors that appear on gauge scale

[ ] (default) | RGB triplet | predefined color name | 1-by-n cell array

Colors that appear on semicircular gauge scale, specified as one of the following:

- An n-by-3 array of RGB values, where each element is a valid RGB value between 0 and 1
- A 1-by-n cell array, where each element is a color name or an RGB triplet. The cell array can contain a mix of RGB triplets and color names.

  MATLAB stores and returns all color names and RGB triplets as an n-by-3 array.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range

`[0,1]`, for example, `[0.4 0.6 0.7]`. This table lists RGB triplet values that have equivalent color strings.

Long Name	Short Name	RGB Triplet
`'yellow'`	`'y'`	`[1 1 0]`
`'magenta'`	`'m'`	`[1 0 1]`
`'cyan'`	`'c'`	`[0 1 1]`
`'red'`	`'r'`	`[1 0 0]`
`'green'`	`'g'`	`[0 1 0]`
`'blue'`	`'b'`	`[0 0 1]`
`'white'`	`'w'`	`[1 1 1]`
`'black`	`'k'`	`[0 0 0]`

Each row of the ScaleColors array represents the color for the *i*th colored scale region.

If you set ScaleColors property without explicitly setting ScaleColorLimits property, then MATLAB determines the ScaleColorLimits property value by creating n equally spaced, nonoverlapping limits that span the semicircular gauge's entire scale. For example, if the Limits property value is `[0,75]`, and the ScaleColors property value is `{'green', 'yellow', 'red'}`, then MATLAB sets ScaleColorLimits to `[0,25; 25,50; 50,75]`

Similarly, if you change Limits, without explicitly setting ScaleColorLimits, then MATLAB sets the ScaleColorLimits property value as described in the preceding paragraph.

Example: `{'blue','green'}`

Example: `{[0,.5,.2],'green'}`

Example: `{[0,.5,.2], [.2,.6,0]}`

`ScaleDirection` — Direction of scale

`'clockwise'` (default) | `'counterclockwise'`

Direction of scale, specified as one of the following:

- `'clockwise'` — The scale appears such that the scale tick values increase in a clockwise manner.
- `'counterclockwise'` — The scale appears such that the scale tick values increase in a counterclockwise manner.

`Size` — Semicircular Gauge size
[120,65] (default) | `[width,height]`

Semicircular Gauge size, specified as [width,height]. The units of measurement are pixels. The Size property value changes if you change the orientation.

`Value` — gauge needle location
0 (default) | numeric

The semicircular gauge needle location, specified as any numeric value. If the value is less than the minimum Limits property value, then the needle points to a location immediately before the beginning of the scale. If the value is more than the maximum Limits property value, then the needle points to a location immediately after the end of the scale.

Changing the Limits property value has no effect on the Value property setting.

Example: `60`

`Visible` — Semicircular Gauge visibility
`true` (default) | `false`

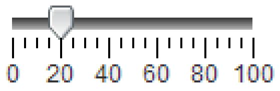
Semicircular Gauge visibility, specified as `true` or `false`. The Visible property determines whether the semicircular gauge is displayed on the screen. If the Visible property of a semicircular gauge is set to `false`, the entire semicircular gauge is hidden, but you can still specify and access its properties.

Slider Properties

Control slider appearance and behavior

Sliders are app components that allow users to select a value along a continuum. Typically, sliders generate an action when an app user changes the value. Properties control the appearance and behavior of a particular instance of a slider. To modify aspects of a slider, change property values. Use dot notation to refer to a particular object and property:

```
slider = uislider;
val = slider.Value;
slider.Value = 20;
```



`Enabled` — Operational state of slider

`true` (default) | `false`

Operational state of slider, specified as `true` or `false`.

If you set this property to `true`, the app user can change the slider value.

If you set this property to `false`, the slider appears dimmed, indicating that the app user cannot change the slider value and the slider will not trigger a callback.

`FontAngle` — Character slant of slider major tick labels

`'normal'` (default) | `'italic'`

Character slant of slider major tick labels, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display slider major tick labels

`'Helvetica'` (default) | string

Font in which to display the slider MajorTickLabels property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of slider major tick labels
12 (default) | positive number

Font size of slider major tick labels, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`Limits` — Minimum and maximum slider values
[0,100] (default) | array

Minimum and maximum slider values, specified as a 2-element numeric array. The first value must be lower than the second value.

`Location` — Location of slider, excluding tick marks and labels, relative to parent
[100,100] (default) | `[x,y]`

Location of slider, excluding tick marks and labels, relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the slider. The units of measurement are pixels.

`MajorTicks` — Major tick mark values
[0, 20, 40, 60, 80, 100] (default) | numeric vector

Major tick mark values, specified as a numeric vector. Any array values that fall outside Limits are not displayed on the slider. MATLAB removes duplicate array values.

Setting the MajorTicks property value sets the MajorTicksMode property value to `'manual'`.

`MajorTickLabels` — Labels on each major tick mark
`{'0','20','40','60','80','100'}` (default) | cell array of strings

Labels on each major tick mark, specified as a cell array of strings. MATLAB uses each element as the label for the corresponding major tick mark.

Setting MajorTickLabels changes the MajorTickLabelsMode value to `'manual'`.

To create a blank label to a major tick, specify an empty string for that tick's corresponding MajorTickLabels element.

If you specify more MajorTickLabels elements than MajorTicks elements, MATLAB ignores the extra labels.

If you specify fewer MajorTickLabels elements than MajorTicks elements, MATLAB leaves the extra ticks unlabeled.

Example: `{'100','80','60','40','20','0'}`

Example: `{'Min','Max'}`

`MajorTickLabelsMode` — Major tick labels creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB creates a cell array of strings to populate MajorTickLabels. MATLAB transforms each numeric major tick element to a string using `num2str`.
- `'manual'` — You specify the MajorTickLabels cell array of strings.

`MajorTicksMode` — Major tick creation mode

`'auto'` (default) | `'manual'`

Major tick creation mode, specified as one of the following:

- `'auto'` — MATLAB populates MajorTicks by creating several major tick marks in the range specified by the Limits property (inclusive).
- `'manual'` — You specify the MajorTicks value array.

`MinorTicks` — Minor tick mark values

`[0:5:100]` (default) | numeric vector

Minor tick mark values, specified as a numeric vector. Any MinorTicks elements that fall outside Limits are not displayed on the gauge. If a minor tick falls on the same value as a major tick, only the major tick is displayed. MATLAB removes duplicate MinorTicks element values. Setting the MinorTicks property value sets the MinorTicksMode property value to `'manual'`.

`MinorTicksMode` — Minor tick creation mode

`'auto'` (default) | `'manual'`

Minor tick creation mode, specified as `'auto'` or `'manual'`.

When MinorTicksMode is set to `'auto'`, MATLAB determines the placement of minor ticks by finding the largest interval between two consecutive major ticks and dividing that interval by 4. For example, if major ticks are uniformly distributed, then the slider will have 3 minor ticks between every major tick.

`Orientation` — Direction in which slider is displayed

`'horizontal'` (default) | `'vertical'`

Direction in which slider is displayed, specified as `'horizontal'` or `'vertical'`.

`OuterLocation` — Location of slider, including tick marks and state labels, relative to parent

[93, 72] (default) | `[x,y]`

Location of slider, including tick marks and state labels, relative to parent, returned as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the slider. The units of measurement are pixels.

`OuterSize` — Size of Slider, including tick marks and labels

[136,40] (default) | `[width,height]`

Size of the slider, including tick marks and labels, returned as `[width,height]`. The units of measurement are pixels.

`Parent` — Parent container of slider

app window object (default) | panel object | button group object

Parent container of the slider, specified as an app window handle, panel, or button group object.

`Size` — Size of slider, excluding tick marks and labels

[120,6] (default) | `[width,height]`.

Size of the slider, excluding tick marks and state labels, specified as `[width,height]`. The units of measurement are pixels.

`Value` — slider value

0 (default) | numeric

The slider value, specified as a numeric. The numeric must be within the range specified for Limits.

`ValueChangedFcn` — Callback function to execute once when app user changes the slider value

[ ] (default) | function handle | cell array | string

Callback function to execute once when an app user changes the slider value, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

Example: @function

Example: {@function, x}

`ValueChangingFcn` — Callback function to execute as app user changes the slider value

[ ] (default) | function handle | cell array | string

The callback function to execute as the app user changes the slider value, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

When an app user drags the slider, MATLAB generates event data and stores it in the event `Value` property, which you can query.

For example, to display the event value in the MATLAB Command Window, specify this code in the `ValueChangingFcn` callback:

```
disp(event.Value)
```

The callback executes as follows:

- If an app user clicks to move the slider to a new value, the callback executes once.

For example, if the slider is on 1.0, and the user single clicks 1.1, the callback executes and the event value updates once to 1.1.

- If an app user clicks and drags the slider to a new value, the callback executes repeatedly.

  For example, if the slider value is 1.0, and the user clicks, holds, and drags to value 1.1, the callback executes and the event `Value` is updated to 1.1. If the user continues the drag to 1.2, the callback executes again and the event `Value` is updated to 1.2. When the user releases the mouse, the callback does not execute.
- If the Slider Value property changes programmatically, then the callback does not execute.

The number of events sent varies, depending on factors such as the mouse speed, the operating system, and the screen resolution.

Example: @function

Example: {@function, x}

`Visible` — Slider visibility
`true` (default) | `false`

Slider visibility, specified as `true` or `false`. The Visible property determines whether the slider is displayed on the screen. If the Visible property of a slider is set to `false`, the entire slider is hidden, but you can still specify and access its properties.

State Button Properties

Control state button appearance and behavior

State buttons are app components that indicate a logical state. Typically, state buttons generate an action when the state changes. Properties control the appearance and behavior of a particular instance of a state button. To modify aspects of a state button, change property values. Use dot notation to refer to a particular object and property:

```
button = uibutton('state');
selected = button.Value;
button.Value = true;
```

`Enabled` — Operational state of state button

`true` (default) | `false`

Operational state of state button, specified as `true` or `false`.

If you set this property to `true`, the appearance of the state button is normal and an app user can select it.

If you set this property to `false`, the appearance of the state button appears dimmed, indicating that an app user cannot select it.

`FontAngle` — Character slant of state button text

`'normal'` (default) | `'italic'`

Character slant of the state button text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display state button text

`'Helvetica'` (default) | string

Font in which to display the state button Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

Example: `'Arial'`

`FontSize` — Font size of state button text
12 (default) | positive number

Font size of state button text, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

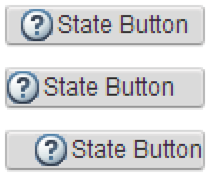
- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`HorizontalAlignment` — Horizontal alignment of text and icon within state button
`'center'` (default) | `'left'` | `'right'`

Horizontal alignment of text and icon within the state button, specified as `'center'`, `'left'`, or `'right'`.

The following image shows the three horizontal alignment options, center, left, and right, when the VerticalAlignment and IconAlignment properties are set to their default values (`'center'` and `'left'`, respectively).



`Icon` — File name of icon to display on button
`''` (default) | string

File name of icon to display on state button, specified as a string.

The string must be the name and extension of an image file on the MATLAB path, or the full path to an image file.

The image file type must be JPEG, GIF, or PNG.

MATLAB clips the image if does not fit within the bounds specified by the state button Size property.

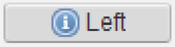
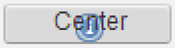
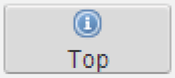
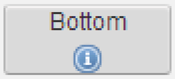
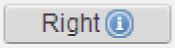
Example: `'icon.png'`

Example: `'C:\Documents\icon.png'`

`IconAlignment` — Location of button icon relative to Text

`'left'` (default) | `'center'` | `'top'` | `'bottom'` | `'right'`

Location of state button icon relative to state button Text (if any), specified as one of the following:

Value	Result
`'left'`	
`'center'`	
`'top'`	
`'bottom'`	
`'right'`	

Note: The preceding images reflect the icon location when HorizontalAlignment and VerticalAlignment are each set to their default value, `'center'`.

If you specify an empty string `('')` for the Text property, the IconAlignment property has no effect.

`Location` — Location of state button relative to parent container

`[100,100]` (default) | `[x,y]`

Location of the state button relative to parent container, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the State Button. The units of measurement are pixels.

`OuterLocation` — Location of state button relative to parent container
`[100,100]` (default) | `[x,y]`

Identical to `Location` for State Buttons.

`OuterSize` — Size of state button
[100,20] (default) | [width, height]

Identical to `Size` for State Buttons.

`Parent` — Parent container of state button
app window object (default) | panel object | button group object

Parent container of state button, specified as an app window, panel, or button group object.

`Size` — State Button size
[100,20] (default) | [width,height]

The state button size, specified as `[width,height]`. The units of measurement are pixels.

`Text` — State Button text
`'State Button'` (default) | string

The state button text, specified as a string.

Example: `'Steady state'`

`Value` — Whether state button is depressed
`true` (default) | `false`

Whether state button is depressed, specified as `true` or `false`.

`ValueChangedFcn` — Callback function that executes when app user presses or releases state button
`[]` (default) | function handle | cell array | string

Callback function that executes when app user presses or releases the state button, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments

- String that is a valid MATLAB expression, which is evaluated in the base workspace.

Example: @function

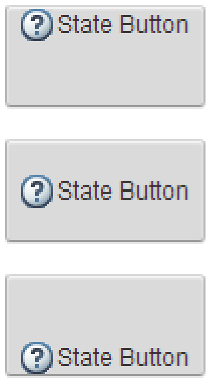
Example: {@function, x}

`VerticalAlignment` — Vertical alignment of text within state button

`'center'` (default) | `'top'` | `'bottom'`

Vertical alignment of text within the state button. specified as `'center'`, `'top'`, or `'bottom'`.

The following image shows the three vertical alignment options when the HorizontalAlignment is `'center'`. In the image, from top to bottom, the vertical alignment values are , `'top'`, `'center'`, `'bottom'`.



Note: Vertical alignment values appear to have the no effect when the button is the default height.

`Visible` — State Button visibility

`true` (default) | `false`

State Button visibility, specified as `true` or `false`. The Visible property determines whether the state button is displayed on the screen. If the Visible property of a state button is set to `false`, the entire state button is hidden, but you can still specify and access its properties.

Switch Properties

Control switch appearance and behavior

Switches are app components that indicate a logical state. Typically, switches generate an action when the state changes. Properties control the appearance and behavior of a particular instance of a switch. To modify aspects of a check box, change property values. Use dot notation to refer to a particular object and property:

```
sliderswitch = uiswitch;
val = sliderswitch.Value;
sliderswitch.Value = true;
```

`Enabled` — Operational state of switch
`true` (default) | `false`

Operational state of switch, specified as `true` or `false`.

If you set this property to `true`, the app user can slide the switch.

If you set this property to `false`, the switch appears dimmed, indicating that an app user cannot slide the switch and it will not trigger a callback.

`Location` — Location of switch, exclusive of state labels relative to parent
`[121,100]` (default) | `[x,y]`

Location of switch, relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the switch. The units of measurement are pixels.

`Orientation` — Direction in which switch is displayed
`'horizontal'` (default) | `'vertical'`

Direction in which switch is displayed, specified as `'horizontal'` or `'vertical'`.

`OuterLocation` — Location of switch, inclusive of state labels, relative to parent
`[100,100]` (default) | `[x,y]`

Location of switch, inclusive of state labels, relative to parent, returned as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the switch, including state labels. The units of measurement are pixels.

`OuterSize` — Size of switch, inclusive of state labels

`[88,20]` (default) | [width height]

Size of switch, inclusive of state labels, returned as `[width,height]`. The units of measurement are pixels.

`Parent` — Parent container of switch

app window object (default) | panel object | button group object

Parent container of switch, specified as an app window, panel, or button group object.

`Size` — Size of switch, exclusive of state labels

`[45,20]` (default) | `[width,height]`

Size of switch, exclusive of state labels, specified as `[width,height]`. The units of measurement are pixels.

`Text` — Switch state labels

`{'Off', 'On'}` (default) | 1-by–2 cell array strings

Switch state labels, specified as a 1-by-2 cell array of strings.

When the switch Orientation property is set to `'vertical'`, the first element of the array appears on the bottom of the switch. When the Orientation property is set to `'horizontal'`, the first element of the array appears to the left of the switch.

`Value` — State of switch

`false` (default) | `true`

State of switch, specified as one of the following values:

- `false`

  The switch points down when the Orientation property is set to `'vertical'` and points left when the Orientation property is set to `'horizontal'`.
- `true`

The switch points up when the Orientation property is set to `'vertical'` and points right when the Orientation property is set to `'horizontal'`.

`ValueChangedFcn` — Code to execute when app user changes switch state

[ ] (default) | function handle | cell array | string

Code to execute when app user changes switch state, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

The app user can change the switch state by clicking and releasing the mouse button anywhere on the switch (including the state labels), or by clicking on the switch, dragging, and releasing the mouse button while still on the switch.

Example: @function

Example: {@function, x}

`Visible` — Switch visibility

`true` (default) | `false`

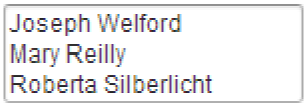
Switch visibility, specified as `true` or `false`. The Visible property determines whether the switch is displayed on the screen. If the Visible property of a switch is set to `false`, the entire switch is hidden, but you can still specify and access its properties.

Text Area Properties

Control text area appearance and behavior

Text areas are app components that enable app users to specify multiple lines of text. Typically, text areas generate an action after an app user enters text. Properties control the appearance and behavior of a particular instance of a text area. To modify aspects of a text area, change property values. Use dot notation to refer to a particular object and property:

```
textarea = uitextarea;
text = textarea.Value;
textarea.Value = {'Joseph Welford';'Mary Reilly';'Roberta Silberlicht'};
```


Joseph Welford
Mary Reilly
Roberta Silberlicht

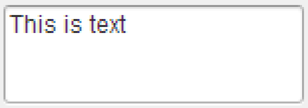
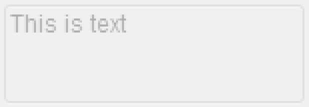
Font Style

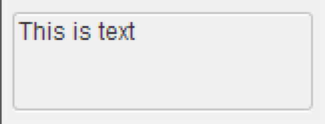
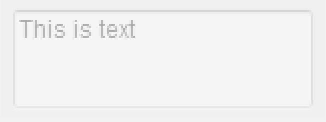
`Editable` — Whether text area is editable

`true` (default) | `false`

Whether the text area is editable, specified as `true` or `false`.

This table summarizes the visual and functional results of various combinations of property values for the Editable and Enabled properties:

Enabled	Editable	Visual Results	Functional Results
`'true'`	`'true'`	 This is text	App user can update the text field. A callback associated with the text field will execute when text changes.
`'false'`	`'false'`	 This is text	App user cannot update the text field.

Enabled	Editable	Visual Results	Functional Results
`'true'`	`'false'`	 This is text	App user cannot update the text field.
`'false'`	`'true'`	 This is text	App user cannot update the text field.

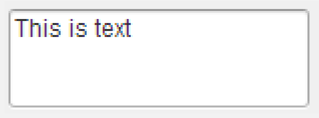
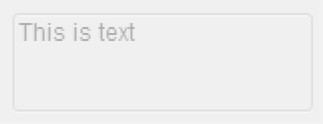
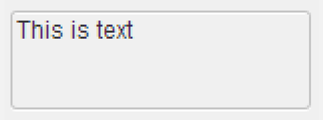
`Enabled` — Operational state of Text Area
`true` (default) | `false`

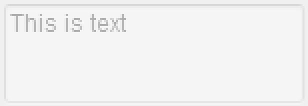
Operational state of Text Area, specified as `true` or `false`.

If you set this property to `true`, the appearance of text in the text area is normal.

If you set this property to `false`, the appearance of text in the text area appears dimmed, indicating that the text area cannot be updated and will not trigger a callback.

This table summarizes the visual and functional results of various combinations of property values for the Editable and Enabled properties:

Enabled	Editable	Visual Results	Functional Results
`'true'`	`'true'`	 This is text	App user can update the text field. A callback associated with the text field will execute when text changes.
`'false'`	`'false'`	 This is text	App user cannot update the text field.
`'true'`	`'false'`	 This is text	App user cannot update the text field.

Enabled	Editable	Visual Results	Functional Results
`'false'`	`'true'`		App user cannot update the text field.

`FontAngle` — Character slant of text area text

`'normal'` (default) | `'italic'`

Character slant of text area text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user's system.

`FontName` — Font in which to display text area text

`'Helvetica'` (default) | string

Font in which to display the text area Value property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

`FontSize` — Font size of text area text

12 (default) | positive number

Font size of text area text, specified as a positive number. The units are points.

Example: 14

`FontWeight` — Thickness of text characters

`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`HorizontalAlignment` — Horizontal alignment of text within text area

`'left'` (default) | `'right'` | `'center'`

Alignment of text within the text area, specified as `'left'`, `'right'`, or `'center'`. The alignment affects the display as the app user edits the text area and how MATLAB displays the text in the app.

`Location` — Location of text area relative to parent
[100,100] (default) | `[x,y]`

Location of text area relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the text area. The units of measurement are pixels.

`OuterLocation` — Location of text area relative to parent container
[100,100] (default) | `[x,y]`

Identical to `Location` for Text Areas.

`OuterSize` — Size of text area, including `Text` value
[150,50] (default) | `[width,height]`

Identical to `Size` for Text Areas.

`Parent` — Parent container of text area
app window object (default) | panel object | button group object

Parent container of text area, specified as an app window, panel, or button group object.

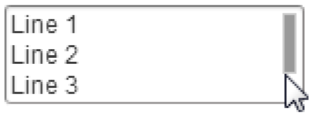
`Size` — Size of text area
[150,50] (default) | `[width,height]`

Size of thetext area, specified as `[width,height]`. The units of measurement are pixels.

`Value` — Text in text area
`{''}` (default) | cell array of strings

Text in text area, specified as a cell array of strings. MATLAB can properly render formatted text, such as this:

```
cellArrayText{1} = sprintf('%s\n%s', 'Line 1', 'Line 2')
cellArrayText{2} = sprintf('%s\n%s', 'Line 3', 'Line 4')
textarea = uitextarea('Value',cellArrayText);
```


Line 1
Line 2
Line 3

If a row of text in the cell array does not fit into the width of the Text Area, MATLAB wraps the text.

If there are too many rows to display in the Text Area size, MATLAB adds a scroll bar.

Example: {'Joseph Welford'; 'Mary Reilly'; 'Roberta Silberlicht'}

`ValueChangedFcn` — Callback to execute after app user commits text entry

`[ ]` (default) | function handle | cell array | string

Callback function to execute after app user enters text, specified as a function handle, a cell array containing a function handle and additional arguments, or a string. The callback executes after the app user types text in the text area and then either presses **Enter**, or clicks outside the text area.

Example: @function

Example: {@function, x}

`Visible` — Text Area visibility

`true` (default) | `false`

Text Area visibility, specified as `true` or `false`. The Visible property determines whether the text area is displayed on the screen. If the Visible property of a text area is set to `false`, the entire text area is hidden, but you can still specify and access its properties.

Toggle Button Properties

Control toggle button appearance

Toggle buttons are app components that appear in a button group, as a set of mutually exclusive options. Typically, the button group to which the toggle button belongs generates an action when the selection changes. Properties control the appearance and behavior of a particular instance of a toggle button. To modify aspects of a toggle button, change property values. Use dot notation to refer to a particular object and property:

```
togbutton = uitogglebutton;
txt = togbutton.Text;
togbutton.Text = 'Histogram';
```

Note: To specify how you want your app to respond when a user changes a toggle button selection, code the ValueChangedFcn callback for the Button Group to which the toggle buttons are parented.

`Enabled` — Operational state of toggle button

`true` (default) | `false`

Operational state of toggle button, specified as `true` or `false`.

If you set this property to `true`, the appearance of the toggle button indicates that an app user can select it.

If you set this property to `false`, the appearance of the toggle button appears dimmed, indicating that an app user cannot select it.

`FontAngle` — Character slant of toggle button text

`'normal'` (default) | `'italic'`

Character slant of the toggle button text, specified as `'normal'` or `'italic'`. Setting this property to `italic` selects a slanted version of the font, if it is available on the app user’s system.

`FontName` — Font in which to display toggle button text

`'Helvetica'` (default) | string

Font in which to display the toggle button Text property value, specified as a string. If the specified font is not available to the app user, MATLAB uses `'Helvetica'`. If `'Helvetica'` is not available, MATLAB uses an available sans serif font.

Example: `'Arial'`

`FontSize` — Font size of toggle button text
12 (default) | positive number

Font size of toggle button text, specified as a positive number. The units of measurement are points.

Example: 14

`FontWeight` — Thickness of text characters
`'normal'` (default) | `'bold'`

Thickness of the text characters, specified as one of these values:

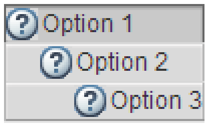
- `'normal'` — Default weight as defined by the particular font
- `'bold'` — Thicker character outlines than `'normal'`

Not all fonts have a bold font weight. Therefore, specifying a bold font weight can result in the normal font weight.

`HorizontalAlignment` — Horizontal alignment of icon and text within toggle button
`'center'` (default) | `'left'` | `'right'`

Horizontal alignment of text and icon within the toggle button, specified as `'center'`, `'left'`, or `'right'`.

The following image shows the three horizontal alignment options, center, left, and right, when the VerticalAlignment and IconAlignment properties are set to their default values (`'center'` and `'left'`, respectively).



`Icon` — File name of icon to display on button
`''` (default) | string

File name of icon to display on toggle button, specified as a string.

The string must be the name and extension of an image file on the MATLAB path, or the full path to an image file.

The image file type must be JPEG, GIF, or PNG.

MATLAB clips the image if does not fit within the bounds specified by the toggle button Size property value.

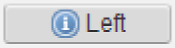
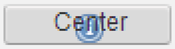
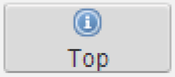
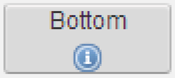
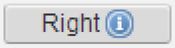
Example: `'icon.png'`

Example: `'C:\Documents\icon.png'`

`IconAlignment` — Location of button icon relative to Text

`'left'` (default) | `'center'` | `'top'` | `'bottom'` | `'right'`

Location of toggle button icon relative to toggle button Text (if any), specified as one of the following:

Value	Result
`'left'`	
`'center'`	
`'top'`	
`'bottom'`	
`'right'`	

Note: The preceding images reflect the icon location when HorizontalAlignment and VerticalAlignment are each set to their default value, `'center'`.

If you specify an empty string (`''`) for the Text property, then the icon aligns as though it is text, according to the values set for HorizontalAlignment and VerticalAlignment. The IconAlignment property value has no effect in this case.

`Location` — Location of toggle button relative to button group
[10,10] (default) | `[x,y]`

Location of the toggle button relative to the parent button group container, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the button group container to the lower-left corner of the Toggle Button. The units of measurement are pixels.

`OuterLocation` — Location of toggle button relative to parent container
[10,10] (default) | `[x y]`

Identical to `Location` for Toggle Buttons.

`OuterSize` — Size of toggle button
[100,20] (default) | [width, height]

Identical to `Size` for Toggle Buttons.

`Parent` — Parent button group container of toggle button
`button group object`

Parent button group container of toggle button, specified as a button group object. See Button Group for details.

`Size` — Toggle Button size
[100,20] (default) | [width, height]

The toggle button size, specified as `[width, height]`. The units of measurement are pixels.

`Text` — Toggle Button text
`'Toggle Button'` (default) | string

The toggle button text, specified as a string.

Example: `'Steady state'`

`Value` — Selection state of toggle button
`true` | `false`

Selection state of toggle button, specified as `true` or `false`.

Within a given toggle button group, only one toggle button can be selected (depressed) at a time.

When the Value property for a toggle button changes to `true`, the Value property for the previously selected toggle button parented to the same button group changes to `false`. In addition, the SelectedObject property value of the button group is updated to reflect the selected toggle button.

If you programmatically change the Value property for a toggle button to `false`, MATLAB sets the first toggle button in the button group to `true`. If the first toggle button is the one for which you set the Value property for a toggle button to `false`, then MATLAB sets the second toggle button in the button group to `true`.

`VerticalAlignment` — Vertical alignment of text within toggle button

`'center'` (default) | `'top'` | `'bottom'`

Vertical alignment of text within the toggle button. specified as `'center'`, `'top'`, or `'bottom'`.

Note: Vertical alignment values appear to have the no effect when the toggle button is the default height.

`Visible` — Toggle Button visibility

`true` (default) | `false`

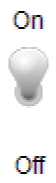
Toggle Button visibility, specified as `true` or `false`. The Visible property determines whether the toggle button is displayed on the screen.

Toggle Switch Properties

Control toggle switch appearance and behavior

Toggle switches are app components that indicate a logical state. Typically, toggle switches generate an action when the state changes. Properties control the appearance and behavior of a particular instance of a toggle switch. To modify aspects of a toggle switch, change property values. Use dot notation to refer to a particular object and property:

```
togswitch = uiswitch('toggle');
val = togswitch.Value;
togswitch.Value = true;
```



`Enabled` — Operational state of toggle switch
`true` (default) | `false`

Operational state of toggle switch, specified as `true` or `false`.

If you set this property to `true`, the end user can toggle the switch.

If you set this property to `false`, the toggle switch appears dimmed, indicating that an app user cannot toggle the switch and it will not trigger a callback.

`Location` — Location of toggle switch, exclusive of state labels, relative to parent
`[121,100]` (default) | `[x,y]`

Location of toggle switch, exclusive of state labels, relative to parent, specified as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the toggle switch. The units of measurement are pixels.

`Orientation` — Direction in which toggle switch is displayed
`'vertical'` (default) | `'horizontal'`

Direction in which toggle switch is displayed, specified as `'horizontal'` or `'vertical'`.

`OuterLocation` — Location of toggle switch, inclusive of state labels, relative to parent
`[100,100]` (default) | `[x,y]`

Location of toggle switch, inclusive of state labels, relative to parent, returned as `[x,y]`, where `x` and `y` specify the coordinates from the lower-left corner of the parent container to the lower-left corner of the toggle switch, including state labels. The units of measurement are pixels.

`OuterSize` — Size of toggle switch, inclusive of state labels
`[20,87]` (default) | [width,height]

Size of toggle switch, inclusive of state labels, returned as `[width,height]`. The units of measurement are pixels.

`Parent` — Parent container of toggle switch
app window object (default) | panel object | button group object

Parent container of toggle switch, specified as an app window, panel, or button group object.

`Size` — Size of toggle switch, exclusive of state labels
`[20,45]` (default) | `[width,height]`

Size of toggle switch, exclusive of state labels, specified as `[width,height]`.

`Text` — Toggle Switch state labels
`{'Off', 'On'}` (default) | 1-by-2 cell array strings

Toggle Switch state labels, specified as a 1-by-2 cell array of strings.

When the toggle switch Orientation property is set to `'vertical'`, the first element of the array appears on the bottom of the toggle switch. When the Orientation property is set to `'horizontal'`, the first element of the array appears to the left of the toggle switch.

`Value` — State of toggle switch
`false` (default) | `true`

State of toggle switch, specified as one of the following values:

- `false`

The toggle switch points down when the Orientation property is set to `'vertical'` and points left when the Orientation property is set to `'horizontal'`.

- `true`

  The toggle switch points up when the Orientation property is set to `'vertical'` and points right when the Orientation property is set to `'horizontal'`.

`ValueChangedFcn` — Code to execute when app user changes toggle switch state

[ ] (default) | function handle | cell array | string

Code to execute when app user changes toggle switch state, specified as one of the following:

- Function handle
- Cell array containing a function handle and additional arguments
- String that is a valid MATLAB expression, which is evaluated in the base workspace.

The app user can change the toggle switch state by clicking and releasing the mouse button anywhere on the toggle switch (including the state labels), or by clicking on the toggle switch, dragging, and releasing the mouse button while still on the toggle switch.

Example: @function

Example: {@function, x}

`Visible` — Toggle Switch visibility

`true` (default) | `false`

Toggle Switch visibility, specified as `true` or `false`. The Visible property determines whether the toggle switch is displayed on the screen. If the Visible property of a toggle switch is set to `false`, the entire toggle switch is hidden, but you can still specify and access its properties.

Function Reference

appdesigner

Create or edit app file in App Designer

Syntax

```
appdesigner
appdesigner file
```

Description

`appdesigner` opens the app building environment, App Designer.

`appdesigner file` opens the specified app file in App Designer.

```
appdesigner myappfile
```

Input Arguments

`file` — Name of app file

string

Name of an app file, specified as a string. If file is overloaded (that is, it appears in multiple folders on the search path), then include a path to the file. The file specification must be the complete or relative path to the file, or the name of a file on the MATLAB path. Otherwise, MATLAB returns an error.

Example: `appdesigner App1`

Examples

Open a New App File

```
appdesigner
```

A new file titled `App1.mlapp` opens in App Designer. `App1.mlapp` does not appear in the MATLAB Current Folder browser until you save the file.

Open an Existing App File

```
appdesigner valueplots
```

App Designer opens and displays the existing `valueplots.mlapp` file, assuming that file is in the current folder or on the MATLAB search path. Otherwise, MATLAB returns a file not found error.

appwindow

Create app window component

Syntax

```
appwindow
window = appwindow(Name,Value)
```

Description

`appwindow` creates a new app window using default property values.

`window = appwindow(Name,Value)` specifies app window properties using one or more `Name,Value` pair arguments.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

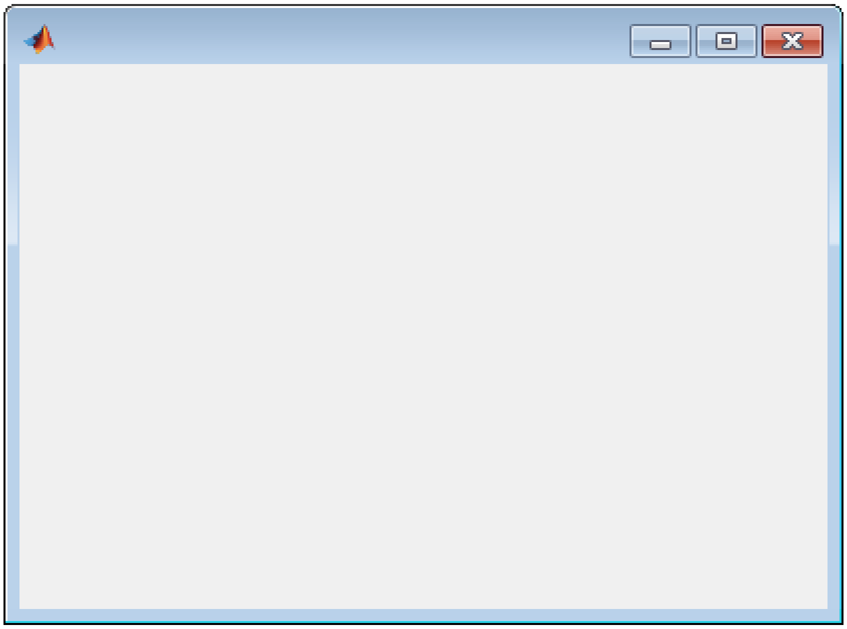
For a list and description of properties (name-value pair arguments), see App Window Properties.

Example: `'Name','Results'`

Examples

Create Default App Window

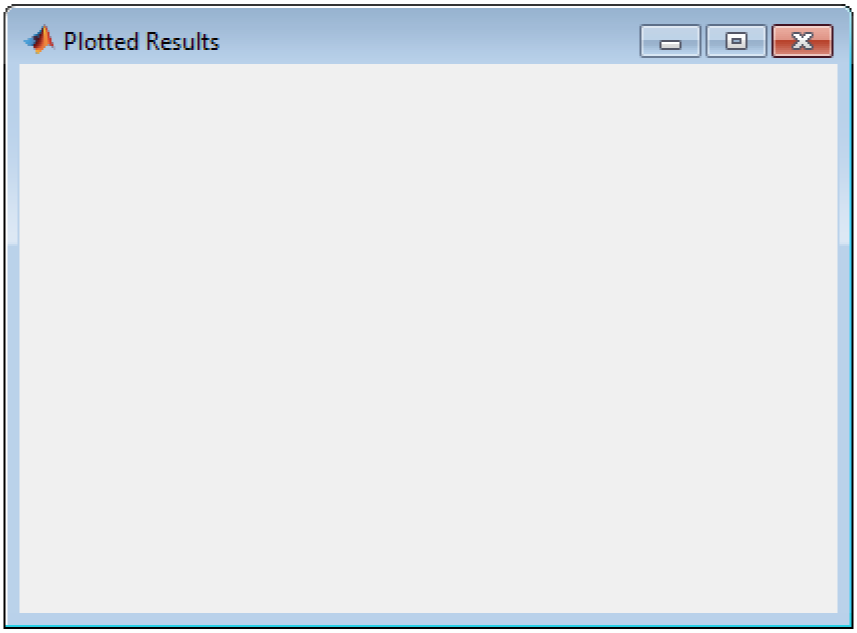
```
appwindow;
```



Access App Window Properties

Create an app window with a specified title.

```
window = appwindow('Name','Plotted Results');
```



Get the app window Position property value.

```
position = window.Position
```

```
ans =

   680   678   560   420
```

See Also

Properties
App Window Properties

axes

Create axes graphics object

Syntax

```
ax = axes('Parent',app)
ax = axes('Parent',app,Name,Value)
```

Description

`ax = axes('Parent',app)` creates an axes object within the specified app window.

`ax = axes('Parent',app,Name,Value)` specifies axes properties using one or more `Name,Value` pair arguments.

Input Arguments

app — app window
app window object

App window, specified as an app window object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a list of properties, see Axes Properties.

Example: `'Color','red'` specifies the color red for the axes back planes.

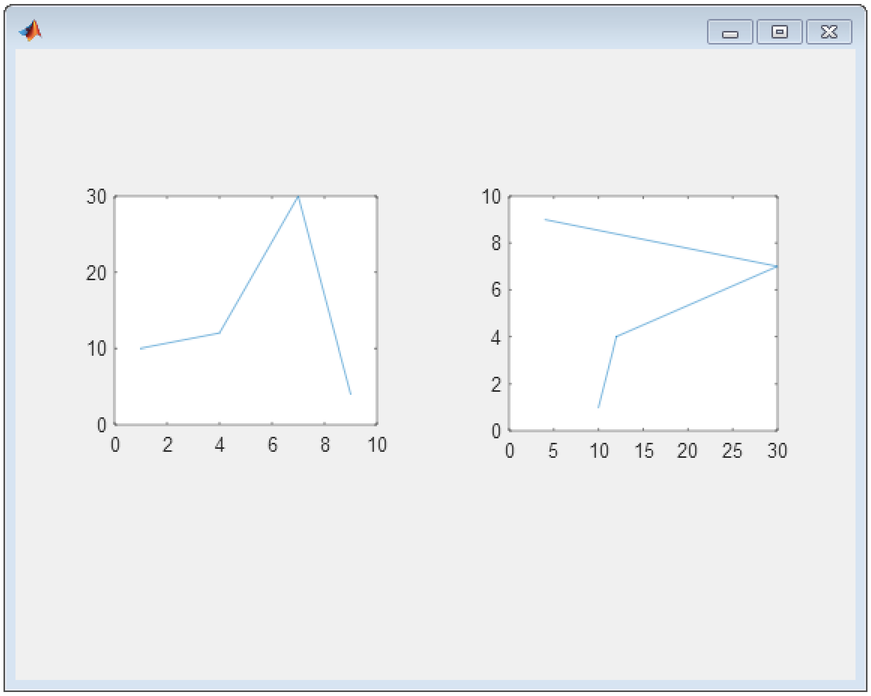
Examples

Multiple Axes in Single App Window

Create and plot into multiple axes in a single app window.

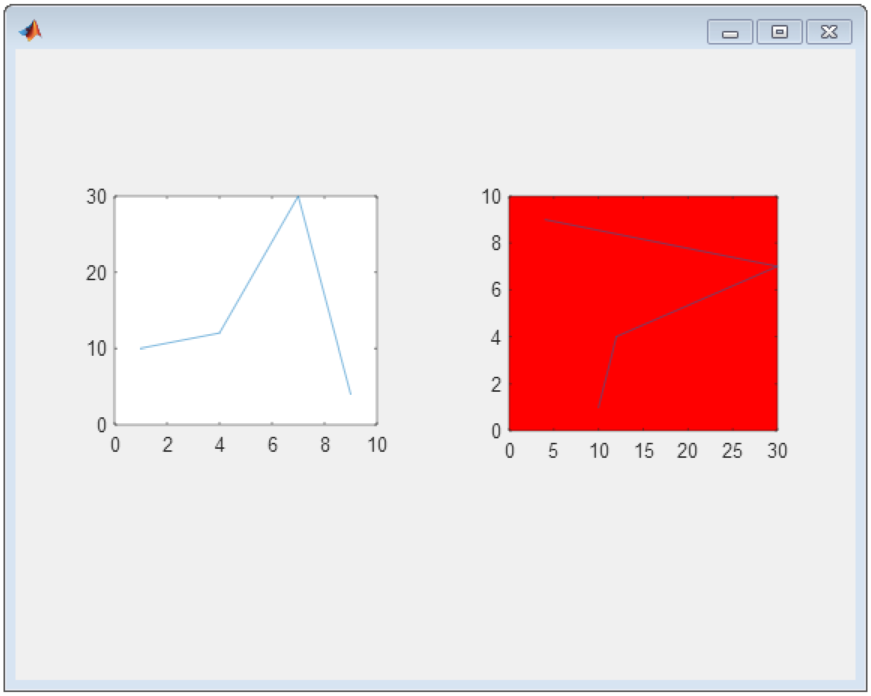
Specify the axes object in the plot command to indicate in which axes you want each plot to be displayed.

```
window = appwindow;
a = [1,4,7,9];
b = [10,12,30,4];
ax1 = axes('Parent',window,'Units','pixels',...
          'Position',[67, 171, 175, 152]);
ax2 = axes('Parent',window,'Units','pixels',...
           'Position',[330,167,179,156]);
plot(ax1,a,b);
plot(ax2,b,a);
```



Change the back plane color of ax2.

```
ax2.Color = 'red';
```



Determine the scale of values along the x-axis in `ax1`.

```
xscale = ax1.XScale
```

```
ans =

linear
```

More About

Tips

- The `axis` (not `axes`) function provides simplified access to commonly used properties that control the scaling and appearance of axes.

- For information on setting default axes properties, see the MATLAB help topic, “Default Property Values”.

See Also

Functions
`axis` | `cla` | `clf` | `gca` | `grid` | `subplot` | `title` | `view` | `xlabel` | `ylabel`

Properties
Axes Properties

uibutton

Create push button or state button component

Syntax

```
button = uibutton
button = uibutton(style)
button = uibutton(parent)
button = uibutton(parent,style)
button = uibutton(___,Name,Value)
```

Description

`button = uibutton` creates a push button in a new app window.

`button = uibutton(style)` creates a button of the specified style.

`button = uibutton(parent)` specifies the object in which to create the button.

`button = uibutton(parent,style)` creates a button of the specified style in the specified parent object.

`button = uibutton(___,Name,Value)` specifies button properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`style` — Type of button
`'push'` (default) | `'state'`

Type of button, specified as one of the following:

- `'push'`

When pressed once, the button appears to depress and release.

- `'state'`

  When pressed once, the button remains in the depressed or released state until it is pressed again.

`Parent` — Parent container of button
app window object (default) | panel object | button group object

Parent container of button, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a style-specific list and description of properties (name-value pair arguments), see Push Button Properties and State Button Properties.

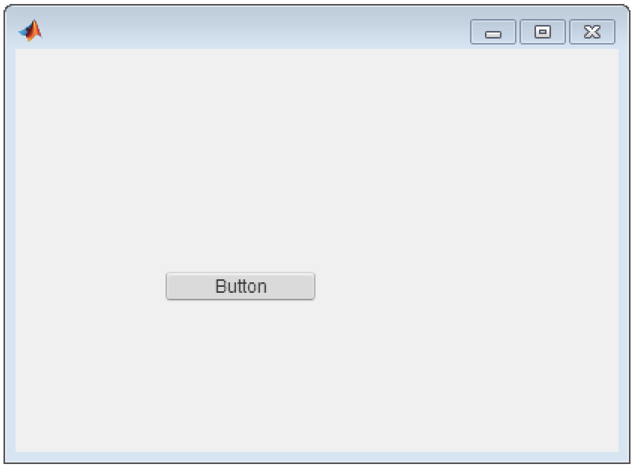
Example: `'Text', 'Start'` specifies that the text, `Start`, is displayed on the button.

Examples

Create a Push Button

Create a push button, the default style for a button.

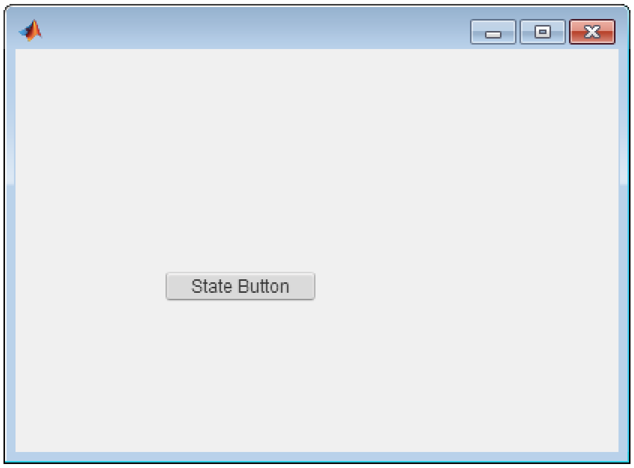
```
button = uibutton;
```



Create a State Button

Create a state button by specifying the style as state.

```
button = uibutton('state');
```



Specify the Parent Object for a Push Button

Specify an app window as the parent object for a push button.

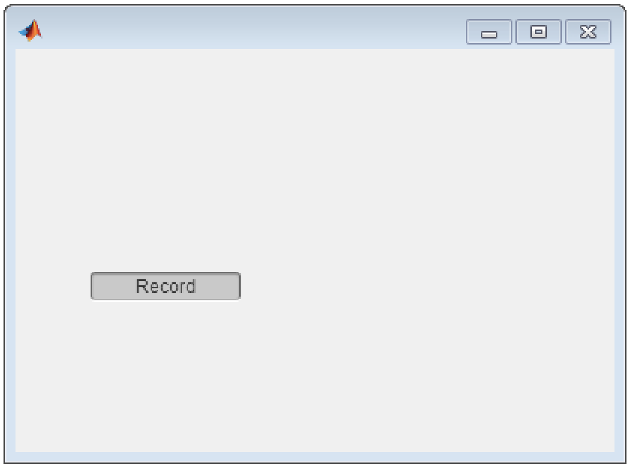
Create two app windows, app1 and app2. Specify app2 as the parent app window for the button.

```
window1 = appwindow('Name','One');
window2 = appwindow('Name','Two');
button = uibutton(window2);
```

Access State Button Property Values

Create a state button and specify property values.

```
button = uibutton('state',...
                  'Text', 'Record',...
                  'Value', true,...
                  'Location',[50,100]);
```



Determine the font size of the state button text:

```
fname = button.FontName
```

```
fname =

Helvetica
```

Change the font of the button text:

```
button.FontName = 'Arial Narrow';
```

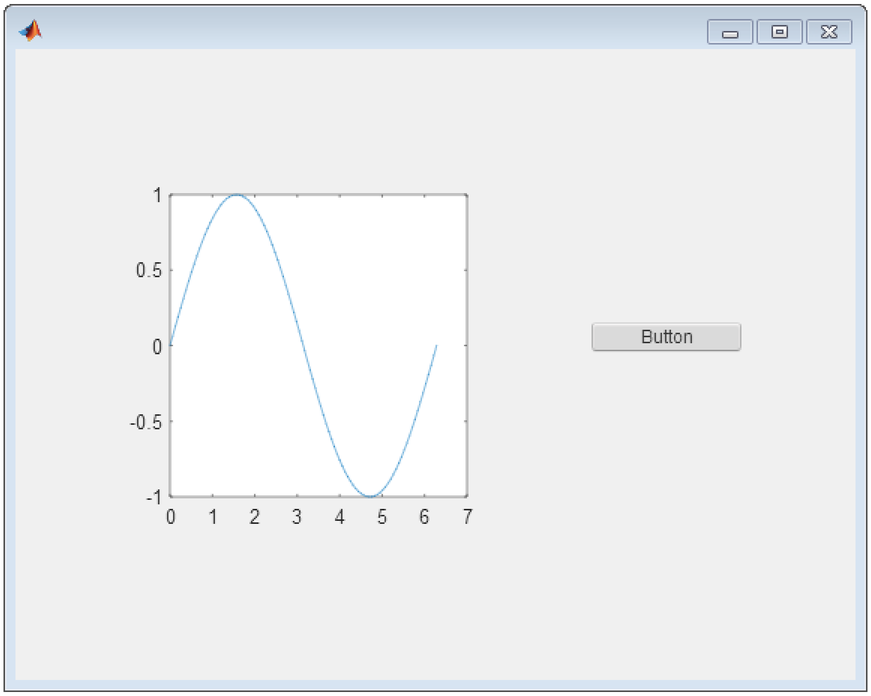
Code the ButtonPushedFcn Callback for a Push Button

Create an app window containing a push button and axes. Code the callback for the push button so that when a user presses the push button, MATLAB plots the specified data in the axes.

Save the following code to a file on your MATLAB path.

```
function buttonPlot
% Create an app window
window = appwindow;

% Create an axes
ax = axes('Parent',window,...
          'Units','pixels',...
          'Position', [104, 123, 198, 201]);

% Create a push button
button = uibutton(window,'push',...
                  'Location',[384, 218],...
                  'ButtonPushedFcn', @(button,event) plotButtonPushed(button,ax));
end

% Create the function for the ButtonPushedFcn callback
function plotButtonPushed(button,ax)
        x = linspace(0,2*pi,100);
        y = sin(x);
        plot(ax,x,y)
end
```

Run `buttonPlot`, and then press the push button.



See Also

Properties

Button Properties | State Button Properties

uicheckbox

Create check box component

Syntax

```
checkbox = uicheckbox
checkbox = uicheckbox(parent)
checkbox = uicheckbox( ___ ,Name,Value)
```

Description

`checkbox = uicheckbox` creates a check box in a new app window.

`checkbox = uicheckbox(parent)` specifies the object in which to create the check box.

`checkbox = uicheckbox( ___ ,Name,Value)` specifies check box properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`parent` — Check box parent container

app window object (default) | panel object | button group object

Check box parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

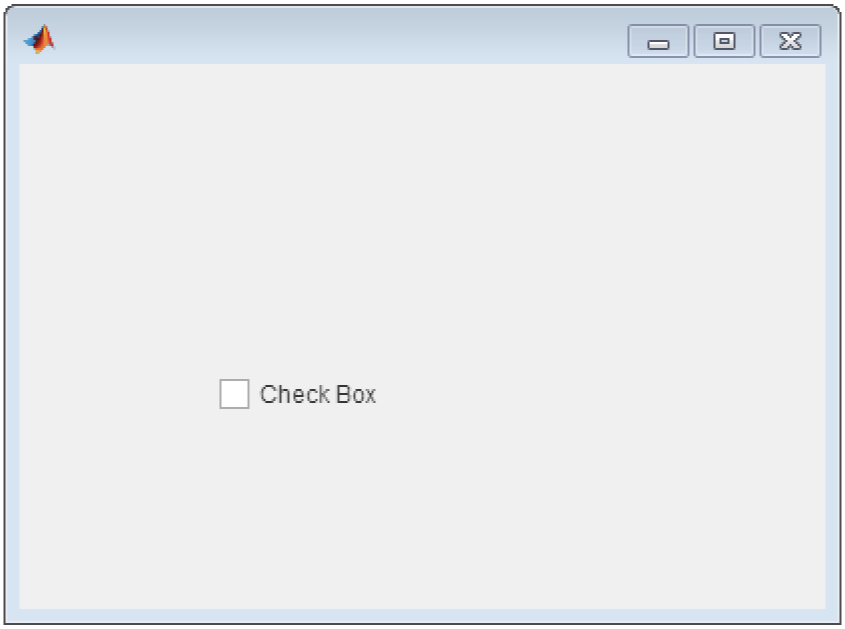
For a list and description of properties (name-value pair arguments), see Check Box Properties.

Example: `'Value',true` specifies that the check box is displayed with a check mark.

Examples

Create Check Box

```
checkbox = uicheckbox;
```



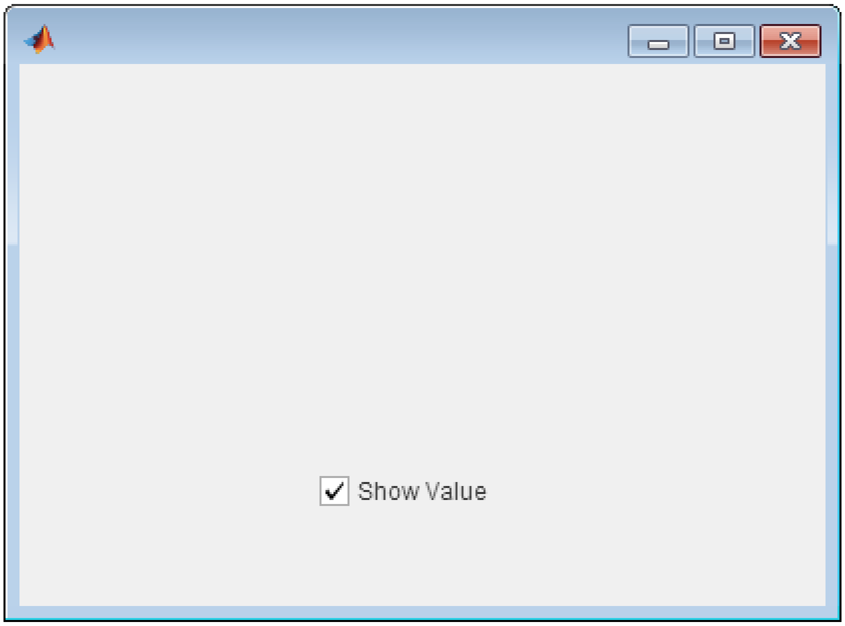
Specify the Parent Object for a Check Box

```
window = appwindow;
checkbox = uicheckbox(window);
```

Access Check Box Property Values

Create a check box and specify property values.

```
checkbox = uicheckbox('Text','Show Value',...
           'Value', true,...
           'Location',[150,50]);
```



Clear the check box:

```
checkbox.Value = false;
```

Determine the font size of the check box text:

```
fsize = checkbox.FontSize
```

```
fsize =

    12
```

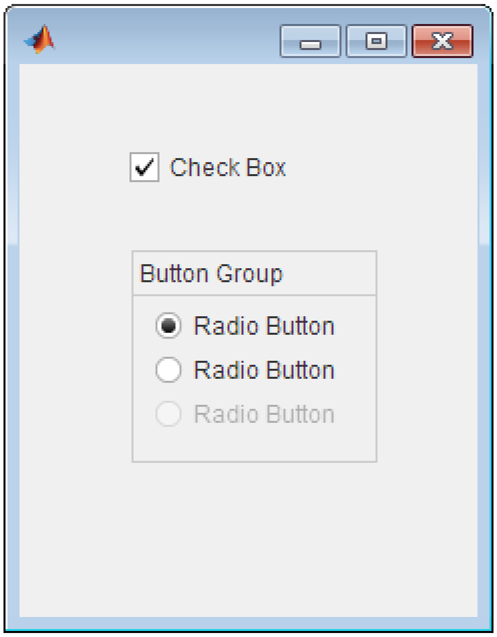
Code the Check Box ValueChangedFcn Callback

Create an app window containing a check box and a button group containing three radio buttons. Code the callback for the check box such that when the user selects the check box, the third button in the button group is disabled. When the user clears the check box, the radio button is enabled again.

Save the following code to a file on your MATLAB path.

```
function disableRadioButton
% Create an app window:
window = appwindow('Position',[100, 100, 229, 276]);



% Create a button group and radio buttons:
buttongroup = matlab.ui.control.ButtonGroup('Parent',window,...
    'Location',[56, 77]);
rb1 = uiradiobutton(buttongroup,'Location',[10,60]);
rb2 = uiradiobutton(buttongroup,'Location',[10,38]);
rb3 = uiradiobutton(buttongroup,'Location',[10,16]);


% Create a checkbox:
uicheckbox(window,'Location',[55, 217],...
    'ValueChangedFcn',@(checkbox,event) cBoxChanged(checkbox,rb3));
end

% Create the function for the ValueChangedFcn callback:
function cBoxChanged(checkbox,rb3)
val = checkbox.Value;
rb3.Enabled = ~val;
end
```

Run `disableRadioButton`, and then clear the check box.



See Also

Properties
Check Box Properties

uidropdown

Create drop down component

Syntax

```
dropdown = uidropdown
dropdown = uidropdown(style)
dropdown = uidropdown(parent)
dropdown = uidropdown(parent,style)
dropdown = uidropdown( ___ ,Name,Value)
```

Description

`dropdown = uidropdown` creates a drop down in a new app window.

`dropdown = uidropdown(style)` creates a drop down of the specified style.

`dropdown = uidropdown(parent)` specifies the object in which to create the drop down.

`dropdown = uidropdown(parent,style)` creates a drop down of the specified style in the specified parent object.

`dropdown = uidropdown( ___ ,Name,Value)` specifies drop down properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`style` — Type of drop down

`'dropdown'` (default) | `'editable'`

Type of drop down, specified as one of the following:

- `'dropdown'`

Enables user to select from a fixed list of options.

- `'editable'`

  Enables user to type text or select from a fixed list of options.

`parent` — Drop down parent container
app window object (default) | panel object | button group object

Drop down parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

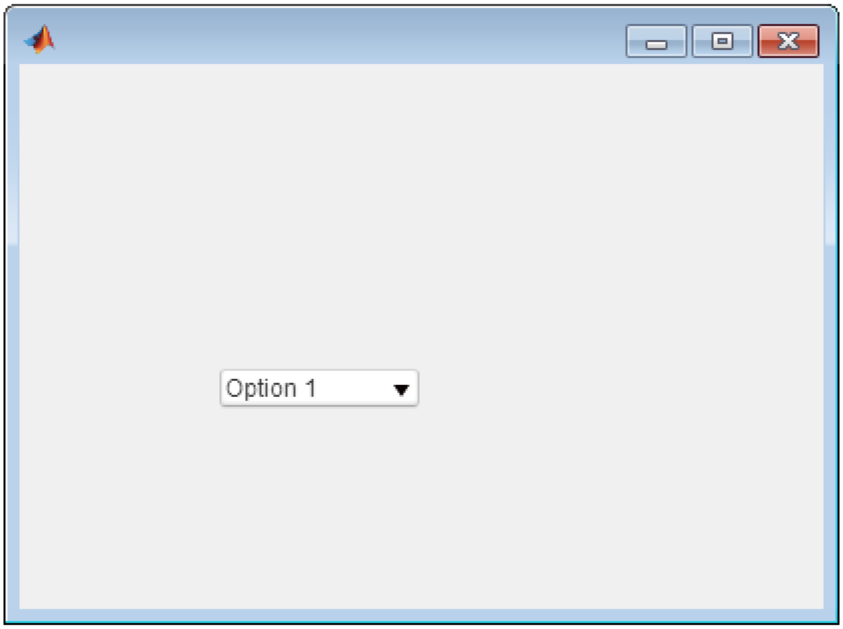
For a style-specific list and description of properties (name-value pair arguments), see Drop Down Properties and Editable Drop Down Properties.

Example: `'Text',{'Red','Yellow','Blue'}` specifies the options presented in the drop down.

Examples

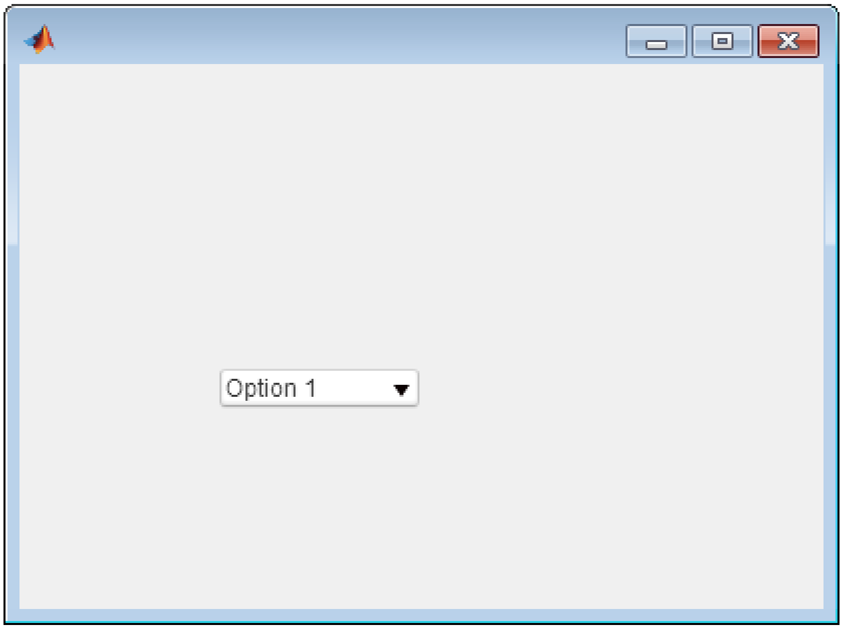
Create a Drop Down

```
dropdown = uidropdown;
```



Create an Editable Drop Down

```
dropdown = uidropdown('editable');
```



Specify the Parent Object for a Drop Down

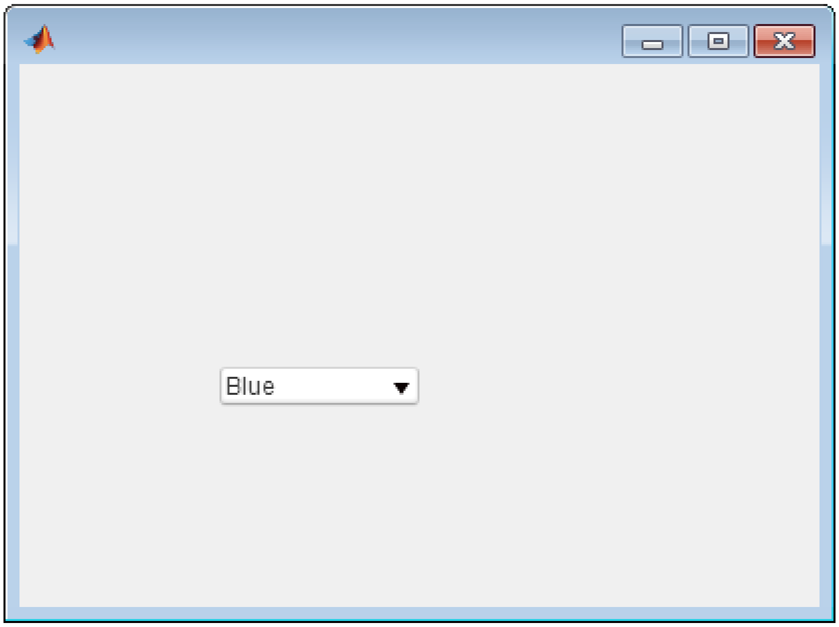
Specify an app window as the parent object for a drop down.

```
window = appwindow;
dropdown = uidropdown(window);
```

Access Drop Down Properties

Create a drop down and specify property values.

```
dropdown = uidropdown('Text',{'Red','Yellow','Blue','Green'},...
                      'Value', 'Blue');
```



Determine the data value associated with the selected option:

```
vdata = dropdown.ValueData
```

```
vdata =

     3
```

Make the font of the text that is displayed in the drop down italic:

```
dropdown.FontAngle = 'italic';
```

Code the Drop Down ValueChangedFcn Callback

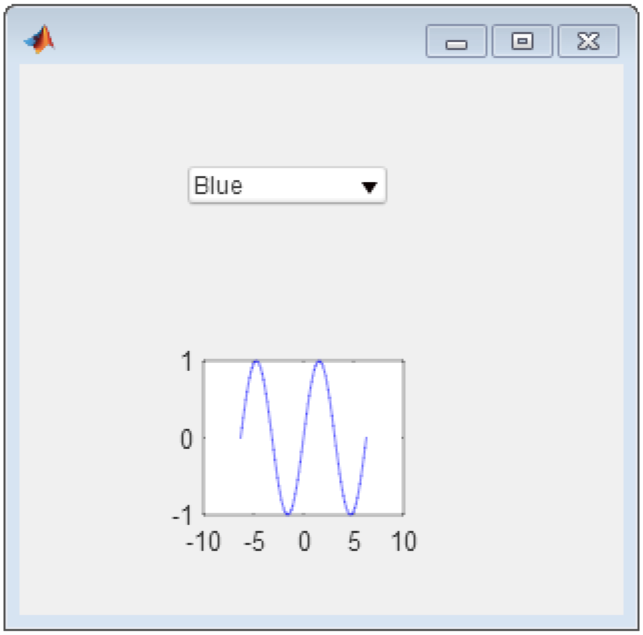
Create an app window containing an axes and a drop down. Code the callback for the drop down such that when the user selects an option, the plot in the axes changes color.

Save the following code to a file on your MATLAB path.

```
function plotOptions
% Create app window and components:
```

```
window = appwindow('Position',[100, 100, 350, 275]);

ax = axes('Parent',window,...
    'Units','pixels',...
    'Position',[93, 51, 100, 77]);

% Create a plot
  x = linspace(-2*pi,2*pi);
  y = sin(x);
  p = plot(ax,x,y);
  p.Color = 'Blue';

dropdown = uidropdown(window,...
    'Location',[84,204],...
    'Text',{'Red','Yellow','Blue','Green'},...
    'Value','Blue',...
    'ValueChangedFcn',@(dropdown,event) selection(dropdown,p));
end

% Create ValueChangedFcn callback:
function selection(dropdown,p)
val = dropdown.Value;
p.Color = val;
end
```

Run `plotOptions`. Select an option from the drop down to change the plot color.



Code the Editable Drop Down ValueChangedFcn Callback Using Value and Data Properties

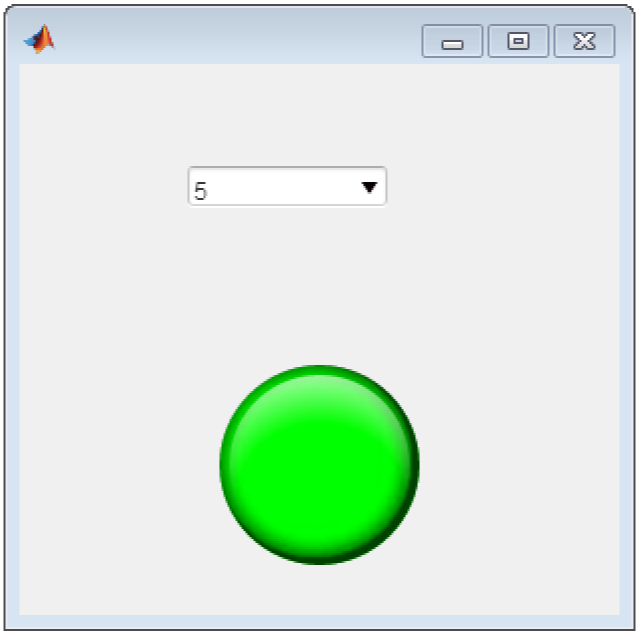
Create an app window containing an editable drop down and a lamp. Code the callback for the drop down such that when the user selects an option or types a numeric value in the drop down, the size of the lamp is multiplied by that value. This example illustrates how to use the Text and Value properties to determine the app user selection.

Save the following code to a file on your MATLAB path.

```
function lampSize
% Create app window and components

window = appwindow('Position',[100, 100, 300, 275]);

lamp = uilamp(window,...
    'Location',[100, 25]);

dropdown = uidropdown(window,'editable',...
    'Location',[84,204],...
    'Text',{'1','2','3','4'},...
    'Value','1',...
    'ValueChangedFcn',@(dropdown,event) optionSelected(dropdown,lamp));
end

% Create ValueChangedFcn callback
    function optionSelected(dropdown,lamp)
        val = dropdown.Value;
        s = [20 20];
        switch val
            case 'Option 1'
                lamp.Size = s;
            case 'Option 2'
                lamp.Size = s.*[2, 2];
            case 'Option 3'
                lamp.Size = s.*[3, 3];
            case 'Option 4'
                lamp.Size = s.*[4, 4];
            otherwise
                v = dropdown.Value;
                m = str2num(v);
                lamp.Size =  s.*[m, m];
        end
    end
```

Run `lampSize` and select various options from the drop down.

Enter a value in the drop down. (If you enter a large value, you might have to resize the app window to see the lamp.)



Code the Editable Drop Down ValueChangedFcn Callback Using ValueData and TextData Properties

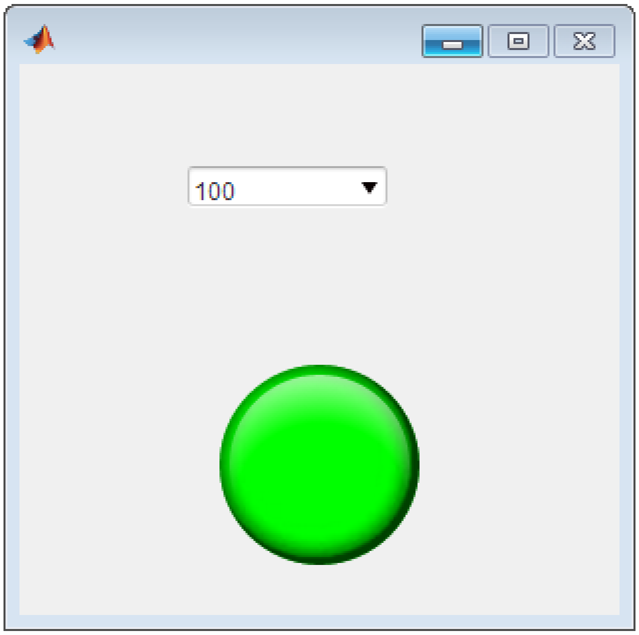
Create an app window containing an editable drop down and a lamp. Code the callback for the drop down such that when the user selects an option the lamp size changes to a size appropriate for the selection. When the user types a numeric value in the drop down, the lamp dimension become that value. This example illustrates how to use the ValueData and TextData properties to determine the app user selection.

Save the following code to a file on your MATLAB path.

```
function lampSizer
% Create app window and components

window = appwindow('Position',[100, 100, 300, 275]);

lamp = uilamp(window,...
    'Location',[100, 25]);

dropdown = uidropdown(window,'editable',...
    'Location',[84,204],...
    'Text',{'small','medium','large','extra large'},...
    'Value','medium',...
    'TextData',{10,20,30,40},...
    'ValueChangedFcn',@(dropdown,event) optionSelected(dropdown,lamp));
end

% Create ValueChangedFcn callback
function optionSelected(dropdown,lamp)
val = dropdown.ValueData;
if (~isempty(val))
    % User selected option
    lamp.Size = val * [1,1];
else
    % User typed a value
    v = dropdown.Value;
    n = str2num(v);
    if (~isempty(n))
        % User typed a number
        lamp.Size = n *[1,1];
    else
        % If user did not type a number,
        % use the default lamp size
        lamp.Size = [20,20];
    end
end
end
```

Run `lampSizer` and select various options from the drop down.

Enter a numeric value in the drop down. (If you enter a large value, you may need to resize the app window to see the lamp.)



See Also

Properties
Drop Down Properties | Editable Drop Down Properties

uieditfield

Create text or numeric edit field component

Syntax

```
editfield = uieditfield
editfield = uieditfield(style)
editfield = uieditfield(parent)
editfield = uieditfield(parent,style)
editfield = uieditfield(___,Name,Value)
```

Description

`editfield = uieditfield` creates a text edit field in a new app window.

`editfield = uieditfield(style)` creates an edit field of the specified style.

`editfield = uieditfield(parent)` specifies the object in which to create the edit field.

`editfield = uieditfield(parent,style)` creates an edit field of the specified style in the specified parent object.

`editfield = uieditfield(___,Name,Value)` specifies edit field properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`style` — Type of edit field
`'text'` (default) | `'numeric'`

Type of edit field, specified as one of the following:

- `'text'`

By default, text edit fields display no text.

- `'numeric'`

  By default, numeric edit fields display the value 0.

`parent` — Edit field parent container
app window object (default) | panel object | button group object

Edit field parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a style-specific list and description of properties (name-value pair arguments), see Edit Field Properties and Numeric Edit Field Properties.

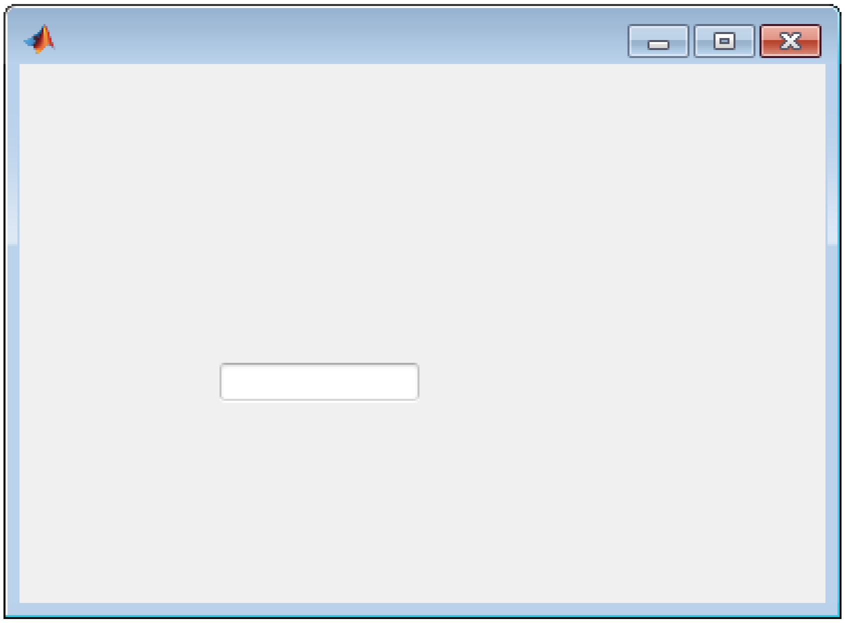
Example: `'Location',[150,150]` specifies the coordinates from the lower-left corner of the parent container to the lower-left corner of the edit field (in pixels).

Examples

Create a Text Edit Field

Create a text edit field, the default style for an edit field.

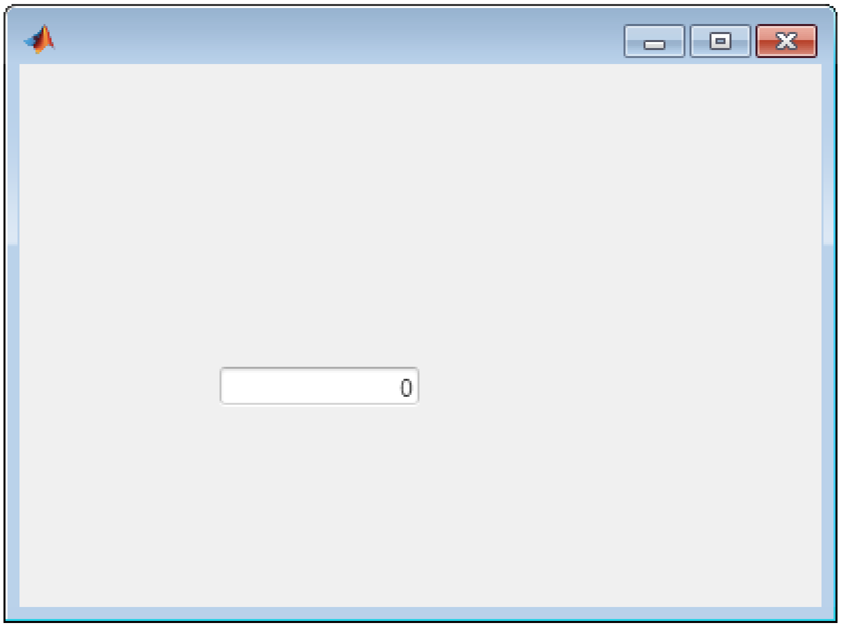
```
editfield = uieditfield;
```



Create a Numeric Edit Field

Create a numeric edit field by specifying the style as numeric.

```
editfield = uieditfield('numeric');
```



Specify the Parent Object for a Numeric Edit Field

Specify an app window as the parent object for a numeric edit field.

```
window = appwindow;
editfield = uieditfield(window,'numeric');
```

Access Numeric Edit Field Property Values

Create a numeric edit field.

```
editfield = uieditfield('numeric');
```

Determine the limits:

```
limits = editfield.Limits
```

```
limits =

  -Inf   Inf
```

The returned values indicate that there are no limits.

Specify the limits as between 0 and 100, exclusive.

```
editfield.Limits = [0,100];
```

Create a Numeric Edit Field and Specify Limit Inclusivity

Create a numeric edit field that allows users to enter a value greater than -5 and less than or equal to 10.

```
editfield = uieditfield('numeric',...
            'Limits', [-5, 10],...
            'LowerLimitInclusivity',false,...
            'UpperLimitInclusivity',true,...
            'Value', 5);
```

Run the command. If you enter a value in the numeric edit field that is outside the limits, MATLAB automatically displays a message indicating the problem and restores the value to the previous valid value.

Create a Numeric Edit Field That Displays Values Using Exactly Two Decimals

Create a numeric edit field that allows users to enter any value, but always displays the value using exactly two decimals. Be aware than MATLAB stores the exact value that the user enters.

```
editfield = uieditfield('numeric',...
            'ValueDisplayFormat', '%.2f');
```

Enter 5.555 in the numeric edit field, and then click away from the edit field. The edit field displays 5.55.

MATLAB stores the original value, 5.555.

Click in the edit field, it displays the value originally typed.

Code the ValueChangedFcn Callback for a Text Edit Field

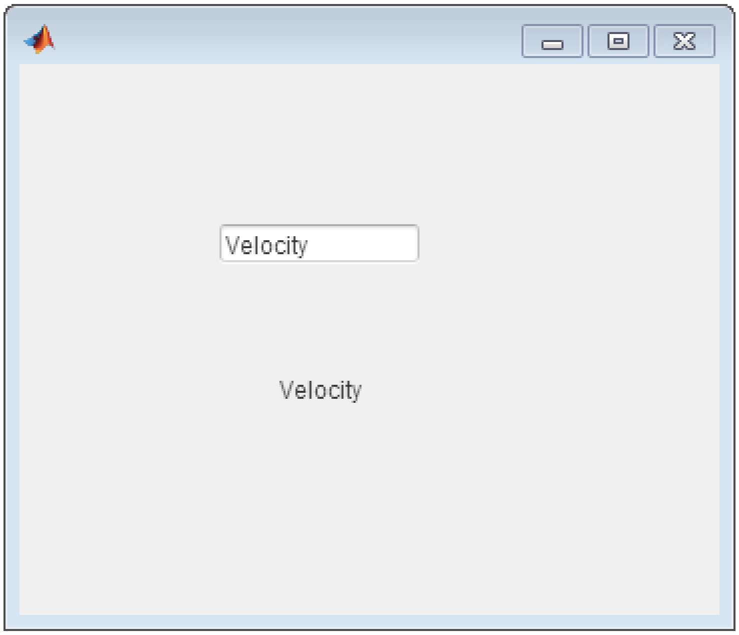
Code the `ValueChangedFcn` callback for a text edit field such that when the user changes text in the edit field, the label is updated to match that text.

Save the following code to a file on your MATLAB path.

```
function textValue
% Create app window and components
```

```
window = appwindow('Position',[100,100,350,275]);

label = uilabel(window,...
    'Location',[130, 100],...
    'Size',[100,20]);

textfield = uieditfield(window,...
    'Location',[100,175],...
    'ValueChangedFcn',@(textfield,event) textChanged(textfield,label));
end

function textChanged(textfield,label)
label.Text = textfield.Value;
end
```

Run `textValue`, and type `Velocity` in the edit field. Click outside the edit field to trigger the callback.



Code the ValueChangedFcn Callback for a Numeric Edit Field

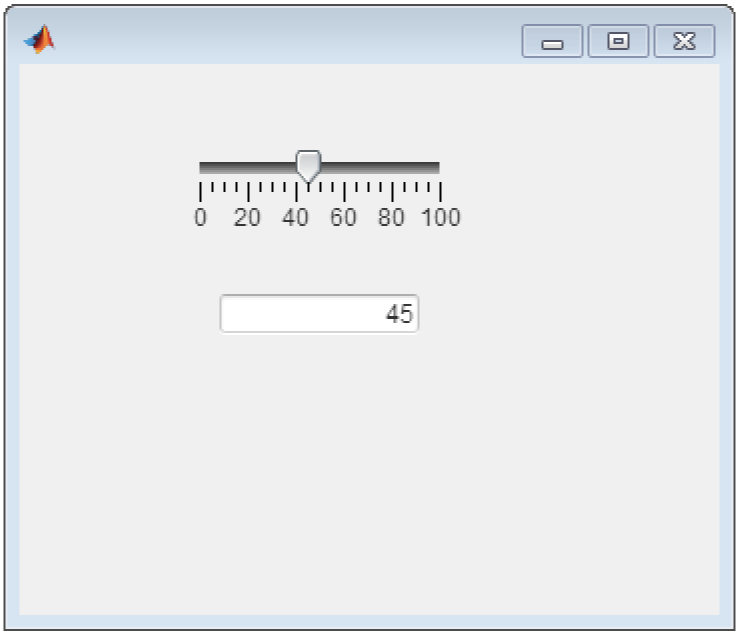
Code the `ValueChangedFcn` callback for an numeric edit field such that when the user changes the value in the edit field, the slider is updated to match that value.

Save the following code to a file on your MATLAB path.

```
function numericEditFieldValue
% Create app window and components

window = appwindow('Position',[100,100,350,275]);

slider = uislider(window,...
    'Location',[90, 220]);

numericfield = uieditfield(window,'numeric',...
    'Location',[100,140],...
    'ValueChangedFcn',@(numericfield,event) numberChanged(numericfield,slider));

end

% Create ValueChangedFcn callback
function numberChanged(numericfield,slider)
slider.Value = numericfield.Value;
end
```

Run `numericEditFieldValue`.

Enter a value between 0 and 100 in the numeric edit field and click outside the field. The slider moves to indicate the numeric edit field value.



See Also

Functions
`uitextarea`

Properties
Edit Field Properties | Numeric Edit Field Properties

uigauge

Create circular, linear, ninety-degree, or semicircular gauge component

Syntax

```
gauge = uigauge
gauge = uigauge(style)
gauge = uigauge(parent)
gauge = uigauge(parent,style)
gauge = uigauge(___,Name,Value)
```

Description

`gauge = uigauge` creates a circular gauge in a new app window.

`gauge = uigauge(style)` specifies the gauge style.

`gauge = uigauge(parent)` specifies the object in which to create the gauge.

`gauge = uigauge(parent,style)` creates a gauge of the specified style in the specified parent object.

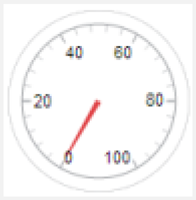
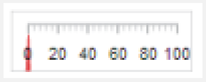
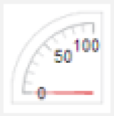
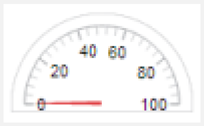
`gauge = uigauge(___,Name,Value)` specifies gauge properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`style` — Type of gauge

`'circular'` (default) | `'linear'` | `'ninetydegree'` | `'semicircular'`

Type of gauge, specified as any one of the following values:

Style	Default Visual	Description
`'circular'`		Component for displaying value over circular span. It always sweeps from 225 degrees in a clockwise direction to 315 degrees.
`'linear'`		Component for displaying value over a linear span.
`'ninetydegree'`		Component for displaying value over a 90 degree span.
`'semicircular'`		Component for displaying value over a semicircular span. The sweep angle is fixed at 180 degrees.

`parent` — Gauge parent container
app window object (default) | panel object | button group object

Gauge parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

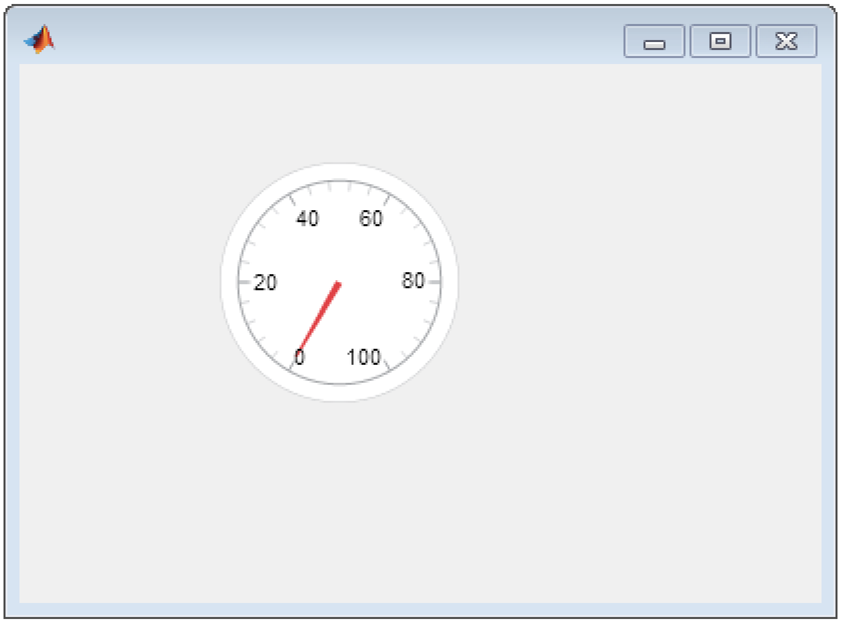
For a style-specific list and description of properties (name-value pair arguments), see Gauge Properties, Semicircular Gauge Properties, Ninety Degree Gauge Properties, and Linear Gauge Properties.

Example: `'Value',150` specifies that the gauge needle points to 150.

Examples

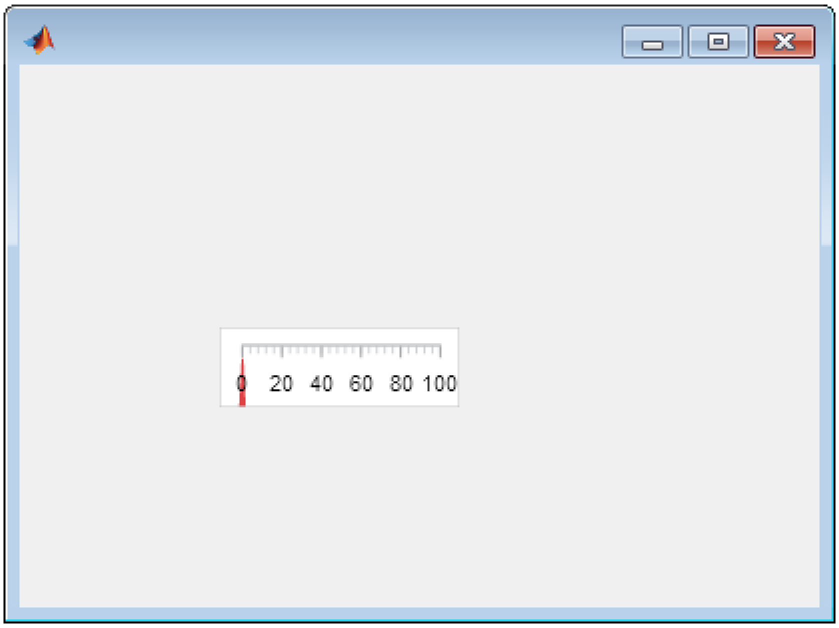
Create a Circular Gauge

```
gauge = uigauge;
```



Create a Linear Gauge

```
gauge = uigauge('linear');
```



Specify the Parent Object for a Gauge

Specify an app window as the parent object for a linear gauge.

```
window = appwindow;
gauge = uigauge(window,'linear');
```

Access Gauge Property Values

Create a circular gauge:

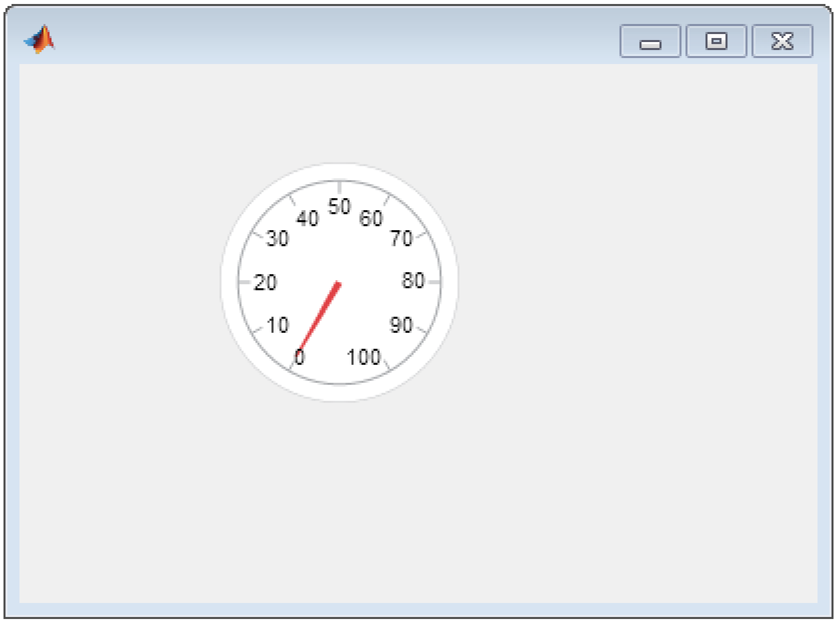
```
gauge = uigauge;
```

Change gauge major ticks, specify matching tick labels, and remove minor ticks.

```
gauge.MajorTicks = [0:10:100];
gauge.MajorTickLabels = {'0','10','20','30','40','50','60','70','80','90','100'};
gauge.MinorTicks = [];
```

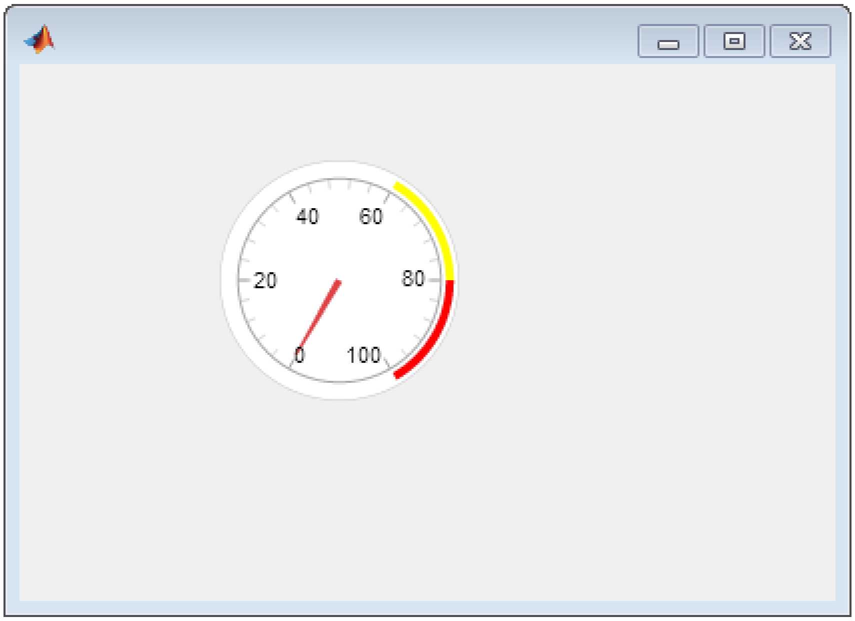
Determine the gauge size:

```
size = gauge.Size
size =

   120   120
```



Create a Circular Gauge, Specifying Scale Colors and Scale Color Limits

```
gauge = uigauge('ScaleColors', {'yellow', 'red'},...
                'ScaleColorLimits', [60,80; 80,100]);
```



See Also

Properties

Gauge Properties | Linear Gauge Properties | Ninety-Degree Gauge Properties | Semicircular Gauge Properties

uiknob

Create knob or discrete knob app component

Syntax

```
knob = uiknob
knob = uiknob(style)
knob = uiknob(parent)
knob = uiknob(parent,style)
knob = uiknob( ___ ,Name,Value)
```

Description

`knob = uiknob` creates a knob in a new app window.

`knob = uiknob(style)` specifies the knob style.

`knob = uiknob(parent)` specifies the object in which to create the knob.

`knob = uiknob(parent,style)` creates a knob of the specified style in the specified parent object.

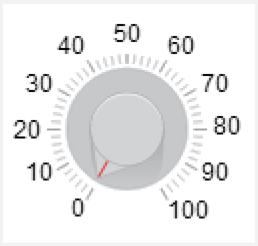
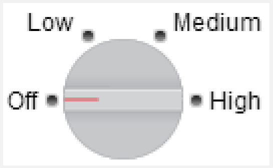
`knob = uiknob( ___ ,Name,Value)` specifies knob properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`style` — Type of knob

`'continuous'` (default) | `'discrete'`

Type of knob, specified as one of the following values:

Style	Default Visual
`'continuous'`	 0 10 20 30 40 50 60 70 80 90 100
`'discrete'`	

`parent` — Knob box parent container
app window object (default) | panel object | button group object

Knob parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a style-specific list and description of properties, see Knob Properties and Discrete Knob Properties.

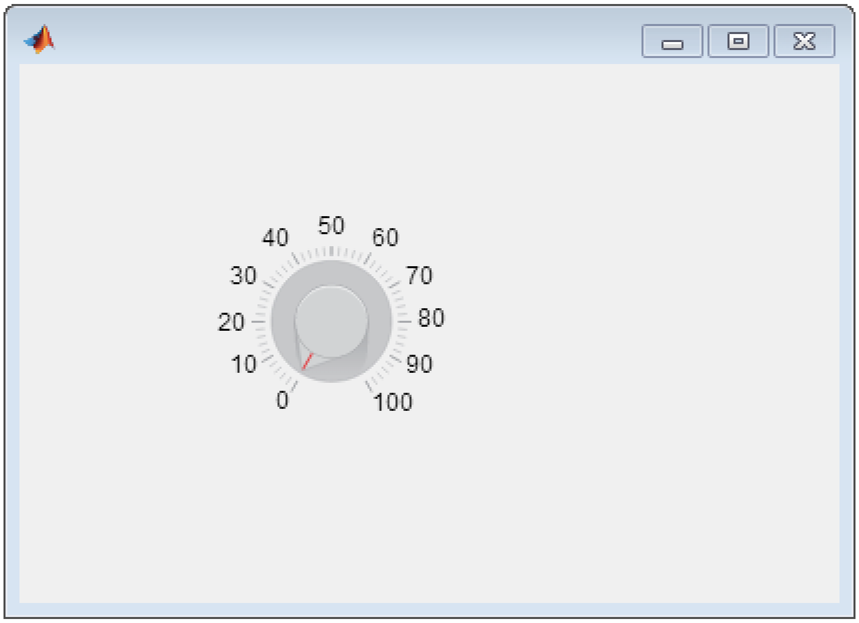
Example: `'Enabled',false` specifies that the knob is displayed dimmed and is not operational.

Examples

Create a Default Knob

Create a continuous knob, the default knob style.

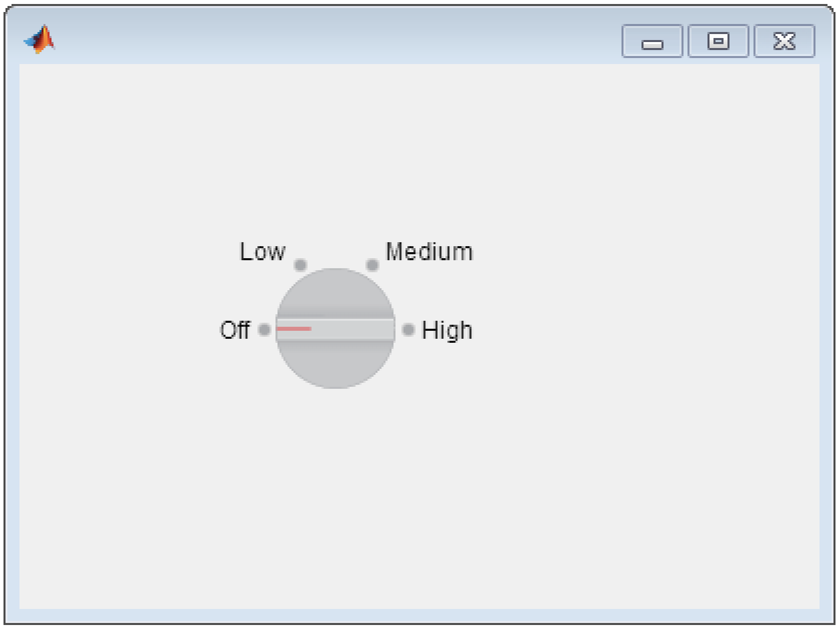
```
knob = uiknob;
```



Create a Discrete Knob

Create a discrete knob by specifying the style as discrete.

```
knob = uiknob('discrete');
```



Specify the Parent Object for a Discrete Knob

Specify an app window as the parent object for a discrete knob.

```
window = appwindow;
knob = uiknob(window,'discrete');
```

Access Continuous Knob Property Values

Create a continuous knob.

```
knob = uiknob;
```

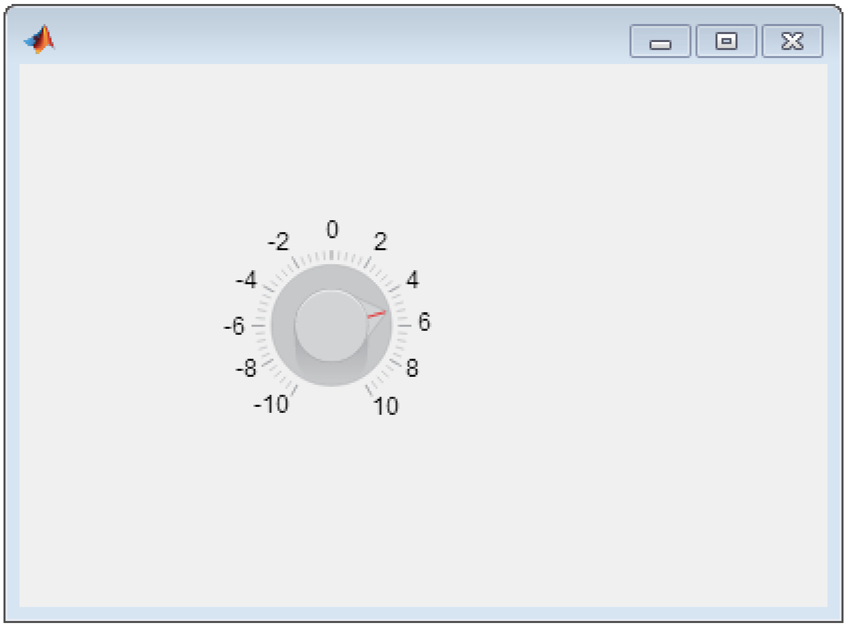
Determine the knob limits.

```
limits = knob.Limits
```

```
limits =

     0   100
```

Change the limits and the value.

```
knob.Limits = [-10,10];
knob.Value = 5;
```



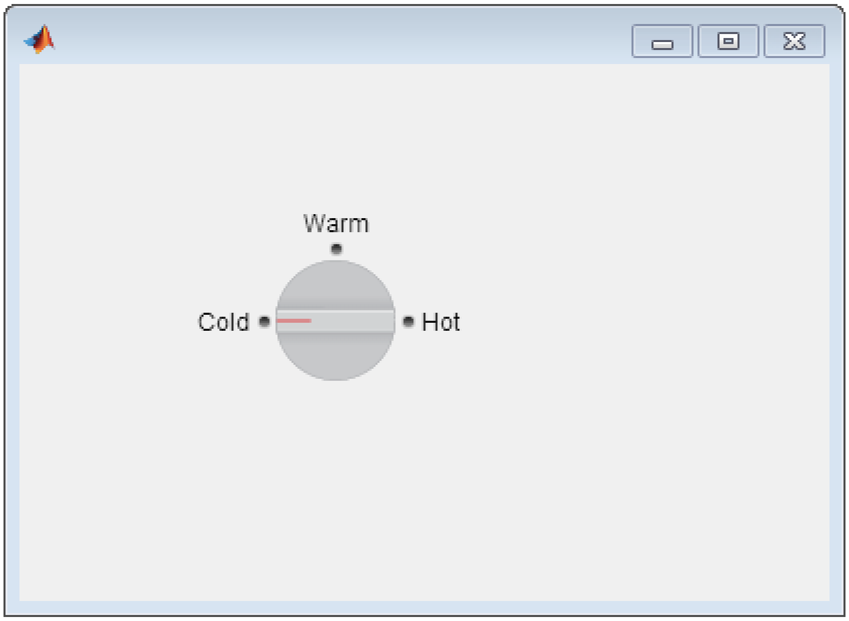
Access Discrete Knob Property Values

Create a discrete knob.

```
knob = uiknob('discrete');
```

Change the knob states. Set the data associated with the knob states to reflect temperatures in degrees Fahrenheit.

```
knob.Text = {'Cold', 'Warm', 'Hot'};
knob.TextData = {32, 80, 212};
```



Get the data value (temperature) associated with the current knob value.

```
degrees = knob.ValueData
```

```
degrees =

    32
```

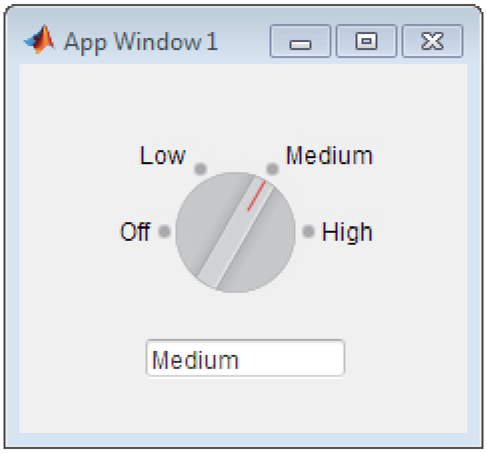
Code the ValueChangedFcn Callback for a Discrete Knob

Create an app window containing a discrete knob and a text field. Code the `ValueChangedFcn` callback such that when the user changes the knob position, the text field is updated to reflect the user's choice.

Copy and paste the following code into a file named `displayknobvalue.m` on your MATLAB path.

```
function displayKnobValue
% Create app window and components

window = appwindow('Position',[100, 100, 283, 275]);

textfield = uieditfield(window,'text',...
                'Location', [69, 82]);

knob = uiknob(window,'discrete',...
           'Location',[89, 142],...
           'ValueChangedFcn',@(knob,event) knobTurned(knob,textfield));
end

function knobTurned(knob,textfield)
  textfield.Value = knob.Value;
end
```

Run `displayKnobValue`, and then turn the knob. When you release the mouse button, the edit field is updated to reflect the knob value.



Code the ValueChangedFcn Callback for a Continuous Knob

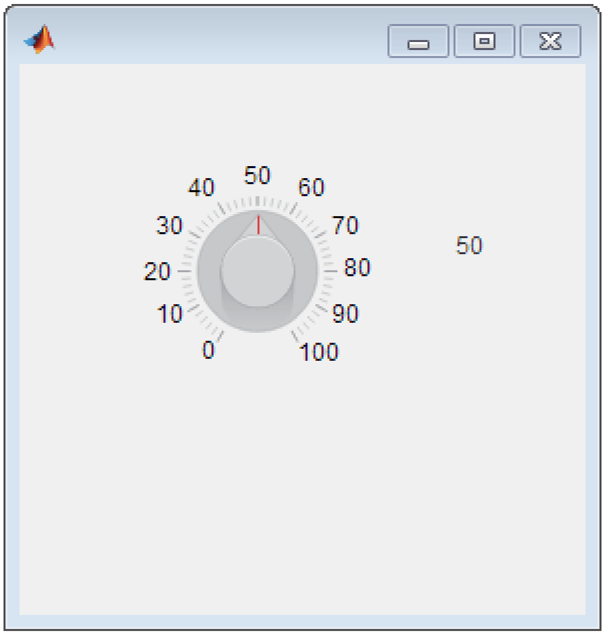
Create an app window containing a continuous knob and a label. Code the `ValueChangedFcn` callback such that when the user changes the knob position, the label is updated to reflect the user's choice.

Copy and paste the following code into a file named `showknobvalue.m` on your MATLAB path.

```
function showKnobValue
% Create app window and components

window = appwindow('Position',[100, 100, 283, 275]);
```

```
label = uilabel(window,'Location',[218,177],...
    'Text','0',...
    'Size', [50,15]);

knob = uiknob(window,...
    'Location',[89, 142],...
    'ValueChangedFcn', @(knob,event) knobTurned(knob,label));
end

% Create ValueChangedFcn callback
function knobTurned(knob,label)
num = knob.Value;
label.Text = num2str(num);
end
```

Run `showKnobValue` and turn the knob. When you release the mouse button, the label is updated to reflect the knob value.



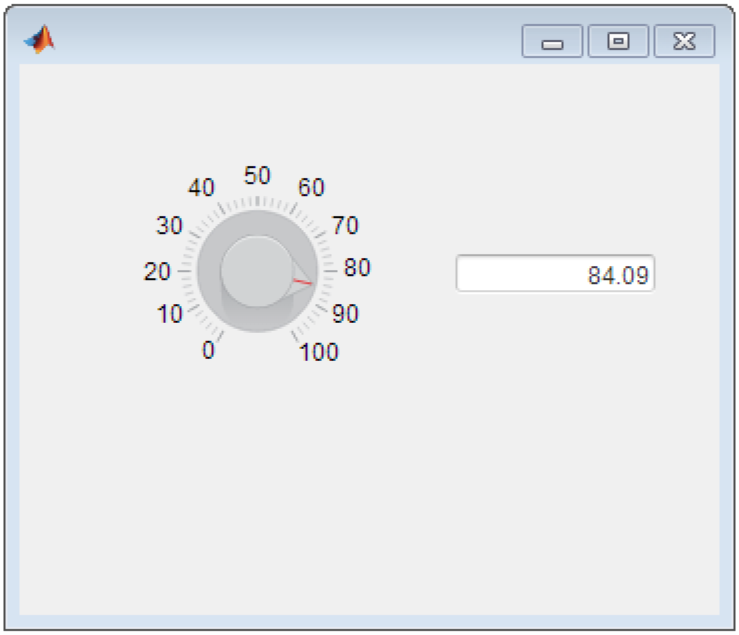
Code the ValueChangingFcn Callback for a Continuous Knob

Create an app window containing a continuous knob and a numeric edit field. Code the knob `ValueChangingFcn` callback such that when the user changes the knob position, the edit field is continuously updated to reflect the knob value. MATLAB stores the changing knob value in the callback event data.

Copy and paste the following code into a file named `showchangingvalue.m` on your MATLAB path.

```
function showChangingValue
% Create app window and components

window = appwindow('Position',[100, 100, 350, 275]);

numberfield = uieditfield(window,'numeric',...
                    'Location',[218,160]);

knob = uiknob(window,...
           'Location',[89, 142],...
           'ValueChangingFcn',@(knob,event) knobTurned(knob,event,numberfield));
end

% Create ValueChangingFcn callback
function knobTurned(knob,event,numberfield)
    numberfield.Value = event.Value;
end
```

Run `showChangingValue`, and turn the knob. As you do so, the numeric edit field is updated to show the changing knob values.



See Also

Properties
Discrete Knob Properties | Knob Properties

uilabel

Create label component

Syntax

```
label = uilabel
label = uilabel(parent)
label = uilabel(___,Name,Value)
```

Description

`label = uilabel` creates a component label with the text `Label`.

`label = uilabel(parent)` specifies the object in which to create the label.

`label = uilabel(___,Name,Value)` specifies label properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`parent` — Label parent container
app window object (default) | panel object | button group object

Label parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a list and description of properties (name-value pair arguments), see Label Properties.

Example: `'Text','Size:'` specifies the label displays the text `Size:`.

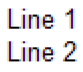
`'Text'` — Label text

`'Label'` (default) | string | cell array of strings

The label text, specified as a string or a cell array of strings. Use a cell array of strings to specify a multiline label.

MATLAB can properly render formatted text, such as this:

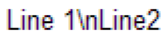
```
text = sprintf('%s\n%s', 'Line 1', 'Line 2')
label = uilabel('Text', text, 'Size',[100,100])
```



Line 1
Line 2

However, MATLAB does not automatically interpret and render text such as this:

```
label = uilabel('Text', 'Line 1\nLine2', 'Size',[100,150])
```



Line 1\nLine2

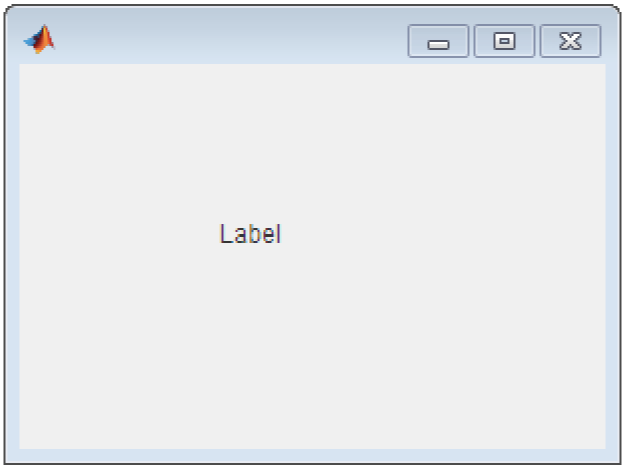
Example: `'Threshold'`

Example: `{'Threshold' 'Value'}`

Examples

Create a Label Component

```
label = uilabel;
```



Specify the Parent Object for a Lamp Component

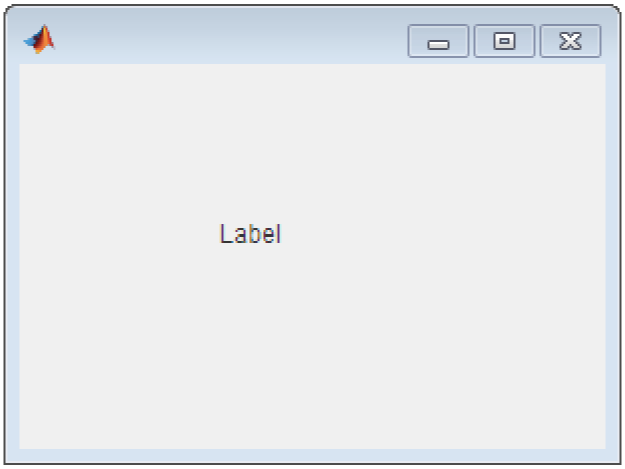
Specify an app window as the parent object for a label.

```
window = appwindow;
label = uilabel(window);
```

Access Label Properties

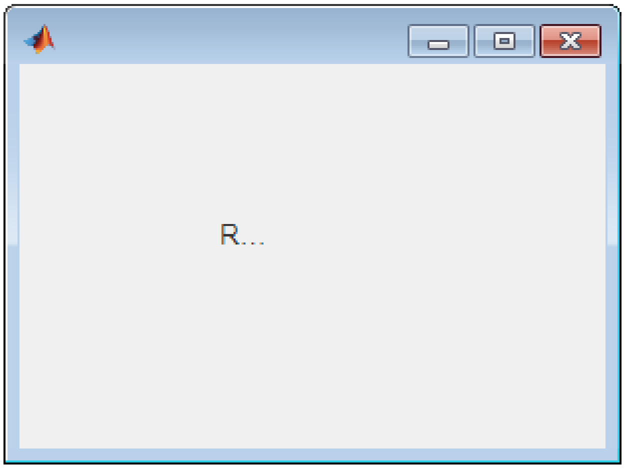
Create a default label.

```
label = uilabel;
```



Change the label text and font size.

```
label.Text = 'Result';
label.FontSize = 14;
```



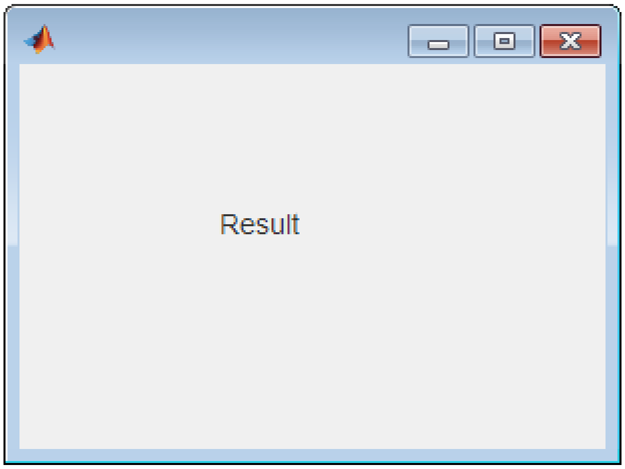
The label is clipped because the current size is not large enough for the changed text.

Determine the current label size.

```
size = label.Size
size =

    31    15
```

Change the label size to accommodate the changed text.

```
label.Size = [60,20];
```



See Also

Properties
Label Properties

uilamp

Create lamp component

Syntax

```
lamp = uilamp
lamp = uilamp(parent)
lamp = uilamp(___,Name,Value)
```

Description

`lamp = uilamp` creates a lamp component in a new app window.

`lamp = uilamp(parent)` specifies the object in which to create the lamp.

`lamp = uilamp(___,Name,Value)` specifies lamp properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`parent` — Lamp parent container
app window object (default) | panel object | button group object

Lamp parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a list and description of properties, see Lamp Properties.

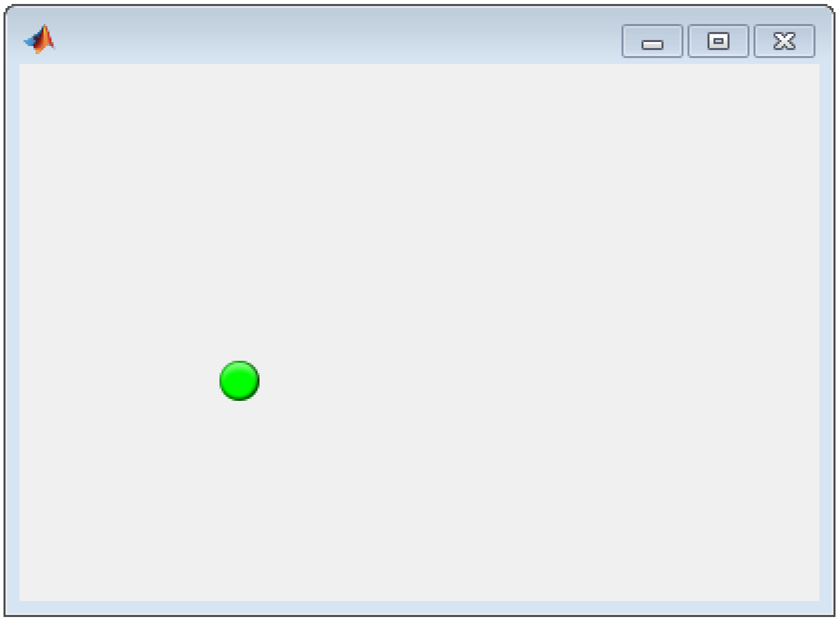
Example: `'Color','red'` specifies that the lamp color is red.

Examples

Create a Lamp Component

Create a green lamp component.

```
lamp = uilamp;
```



Specify the Parent Object for a Lamp Component

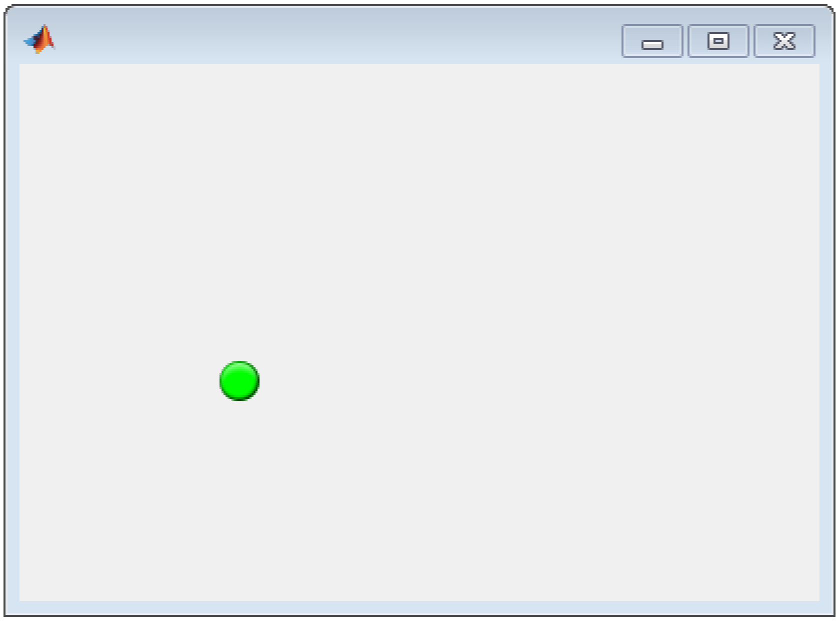
Specify an app window as the parent object for a lamp.

```
window = appwindow;
lamp = uilamp(window);
```

Access Lamp Properties

Create a default lamp.

```
lamp = uilamp;
```



Determine the current color of the lamp.

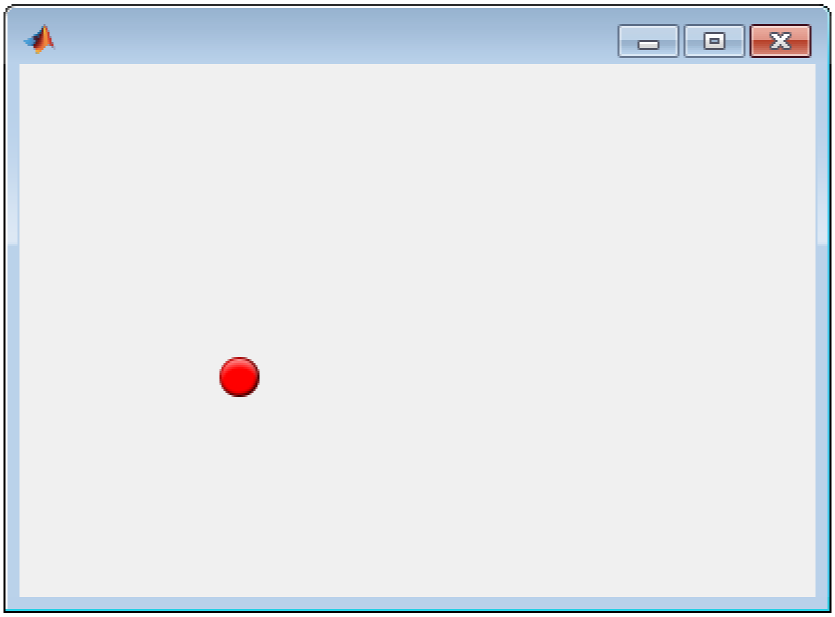
```
color = lamp.Color
```

```
color =

     0     1     0
```

MATLAB returns the RGB value for green.

Change the lamp color to red.

```
lamp.Color = 'red';
```



Change the lamp color to blue.

```
lamp.Color = [0,0,1];
```

See Also

Properties
Lamp Properties

uilistbox

Create list box component

Syntax

```
listbox = uilistbox
listbox = uilistbox(parent)
listbox = uilistbox(___,Name,Value)
```

Description

`listbox = uilistbox` creates a list box in a new app window.

`listbox = uilistbox(parent)` specifies the object in which to create the list box.

`listbox = uilistbox(___,Name,Value)` specifies list box properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`parent` — List box parent container
app window object (default) | panel object | button group object

List box parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

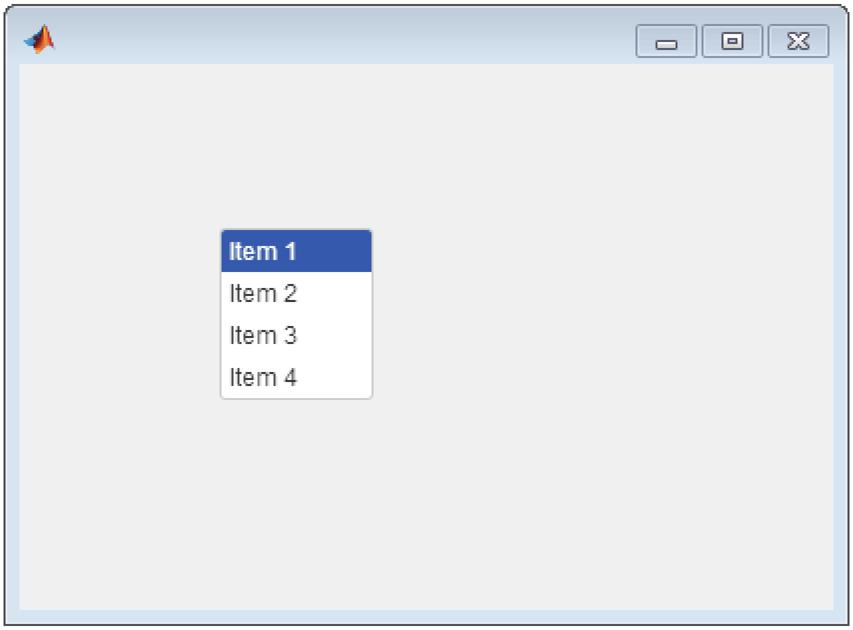
For a list of properties, see List Box Properties.

Example: `'Text',{'Model 1','Model 2', 'Model 3', 'Model 4'}` specifies the list box options that the user sees, from top to bottom.

Examples

Create a List Box

```
listbox = uilistbox;
```



Specify the Parent Object for a List Box

Specify an app window as the parent object for a list box.

```
window = appwindow;
listbox = uilistbox(window);
```

Access List Box Property Values

Create a default list box.

```
listbox = uilistbox;
```

Determine whether the list box allows multiple selections.

```
multi = listbox.MultiSelect
```

```
multi =

     0
```

MATLAB returns zero, indicating that multiselection is not enabled.

Enable multiselection.

```
listbox.MultiSelect = true;
```

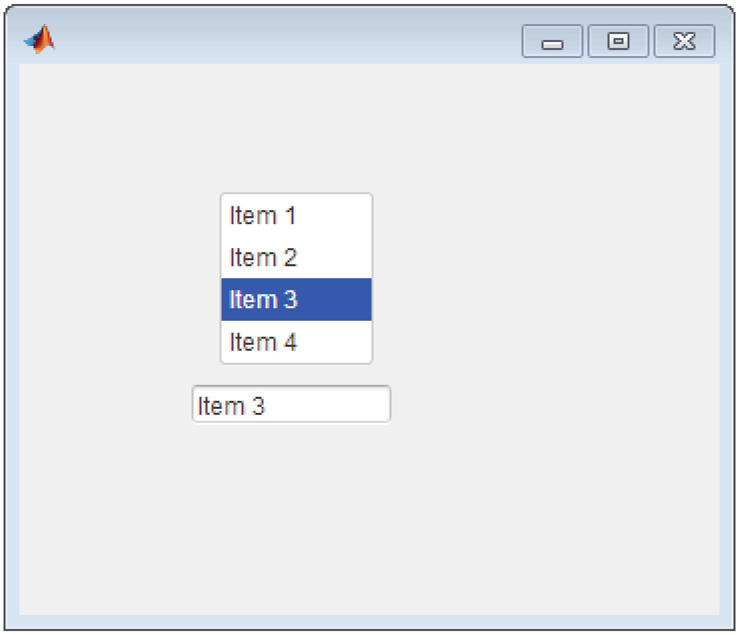
Code the ValueChangedFcn Callback for a List Box to Display Selection

Create an app window containing a list box and a text edit field. Code the callback for the list box so that when a user selects a list option, MATLAB displays that option in the text edit field.

Save the following code to a file on your MATLAB path.

```
function listBoxSelection
% Create app window and components

window = appwindow('Position',[100,100,350,275]);

textfield = uieditfield(window,...
    'Location',[86,90]);
    textfield.Value = 'Item 1';

lbox = uilistbox(window,...
    'Location',[100, 120],...
    'ValueChangedFcn', @(lbox,event) selectionChanged(lbox,textfield));
end

% Create ValueChangedFcn callback
function selectionChanged(lbox,textfield)
textfield.Value = lbox.Value;
end
```

Run `listBoxSelection` and select an option from the list. The text edit field is updated to reflect your choice.



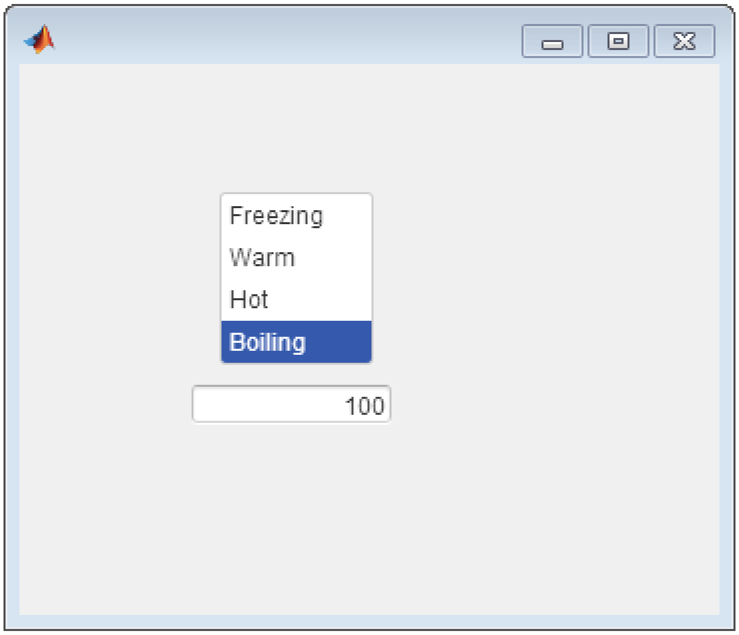
Code the ValueChangedFcn Callback for a List Box to Display Data Associated with Selection

Create an app window containing a list box and a numeric edit field. Code the callback for the list box so that when a user selects a list option, MATLAB displays the data associated with that option in the numeric edit field.

Save the following code to a file on your MATLAB path.

```
function dataSelection
% Create app window and components

window = appwindow('Position',[100,100,350,275]);

numericfield = uieditfield(window,'numeric',...
     'Location',[86,90]);


lbox = uilistbox(window,...
       'Text', {'Freezing', 'Warm', 'Hot', 'Boiling'},...
       'TextData', {0, 25, 40, 100},...
        'Location',[100, 120],...
        'ValueChangedFcn', @(lbox,event) selectionChanged(lbox,numericfield));
end

% Create ValueChangedFcn callback
function selectionChanged(lbox,numericfield)
numericfield.Value = lbox.ValueData;
end
```

Run `dataSelection` and select an option from the list. The numeric edit field is updated to reflect the data associated with your choice.



Code the ValueChangedFcn Callback for a Multiselection List Box

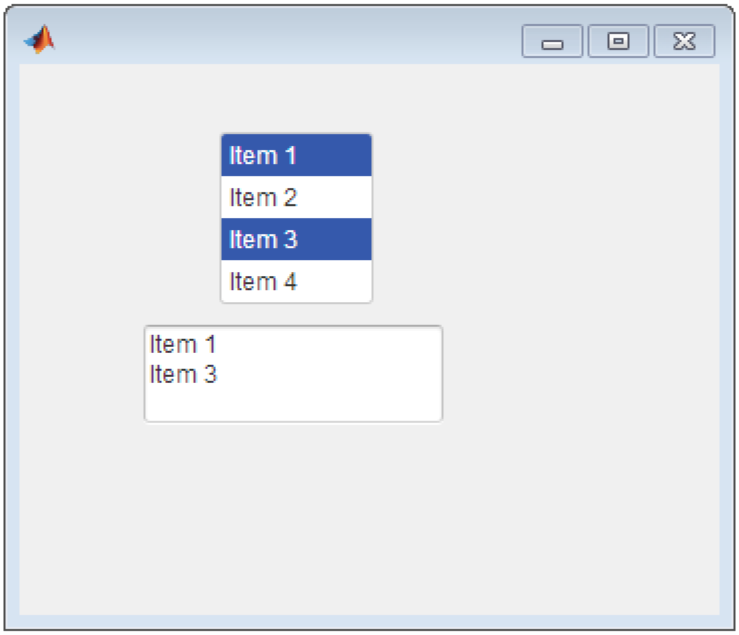
Create an app window containing a list box and a text area field. Code the callback for the list box so that when a user selects one or more list options, MATLAB displays those options in the text area.

Save the following code to a file on your MATLAB path.

```
function multiselect
```

```
% Create app window and components

window = appwindow('Position',[100,100,350,275]);

textarea = uitextarea(window,...
    'Location',[62,90]);

listbox = uilistbox(window,...
    'Location',[100, 150],...
    'Multiselect',true,...
    'ValueChangedFcn',@(listbox,event) selectionChanged(listbox,textarea));

end

% Create ValueChangedFcn callback
function selectionChanged(listbox,textarea)
textarea.Value = listbox.Value;
end
```

Run `multiselect` and select options from the list. The text edit field is updated to reflect your choice.



See Also

Properties
List Box Properties

uiradiobutton

Create radio button component

Syntax

```
radiobutton = uiradiobutton(parent)
radiobutton = uiradiobutton(parent,Name,Value)
```

Description

`radiobutton = uiradiobutton(parent)` creates a radio button within the specified parent button group.

`radiobutton = uiradiobutton(parent,Name,Value)` specifies radio button properties using one or more `Name,Value` pair arguments.

Input Arguments

`parent` — Radio button parent container
`matlab.ui.control.ButtonGroup` object

Radio button parent container, specified as a `matlab.ui.control.ButtonGroup` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Text', 'French'` specifies that the text, `French`, displays next to the radio button.

Examples

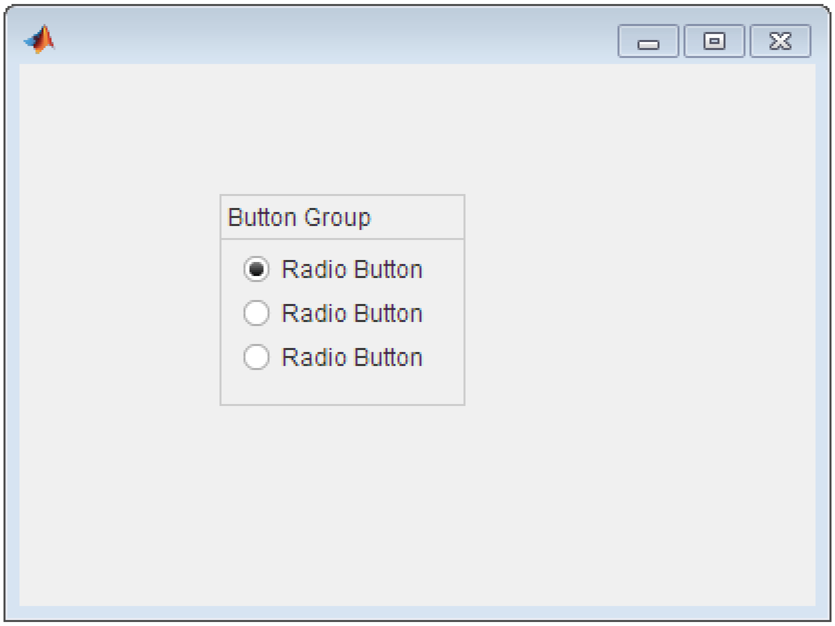
Create Radio Buttons Within a Button Group and Access Property Values

To create a radio button, first create an app window and a button group object.

```
window = appwindow;
bg = matlab.ui.control.ButtonGroup('Parent',window);
```

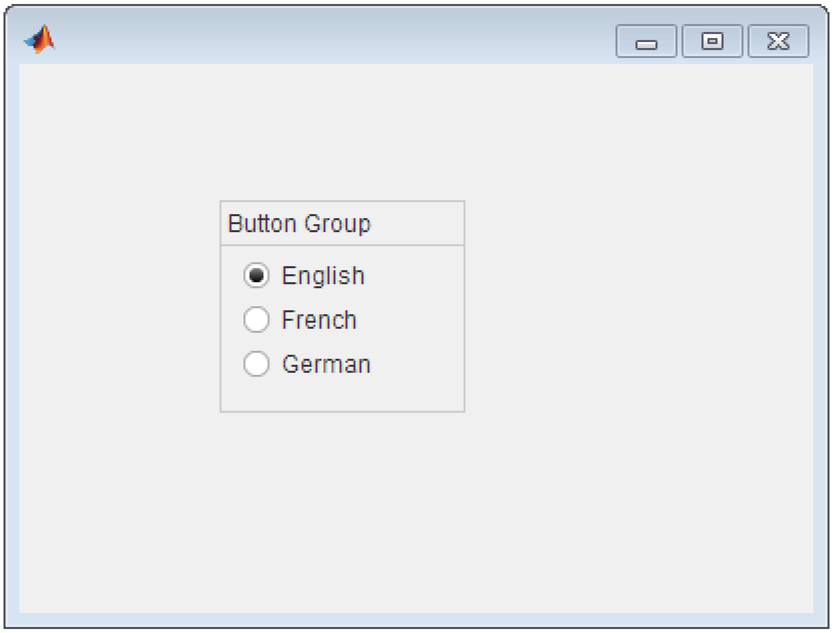
Create three radio buttons and specify the location of each.

```
rb1 = uiradiobutton(bg,'Location',[10,60]);
rb2 = uiradiobutton(bg,'Location',[10,38]);
rb3 = uiradiobutton(bg,'Location',[10,16]);
```



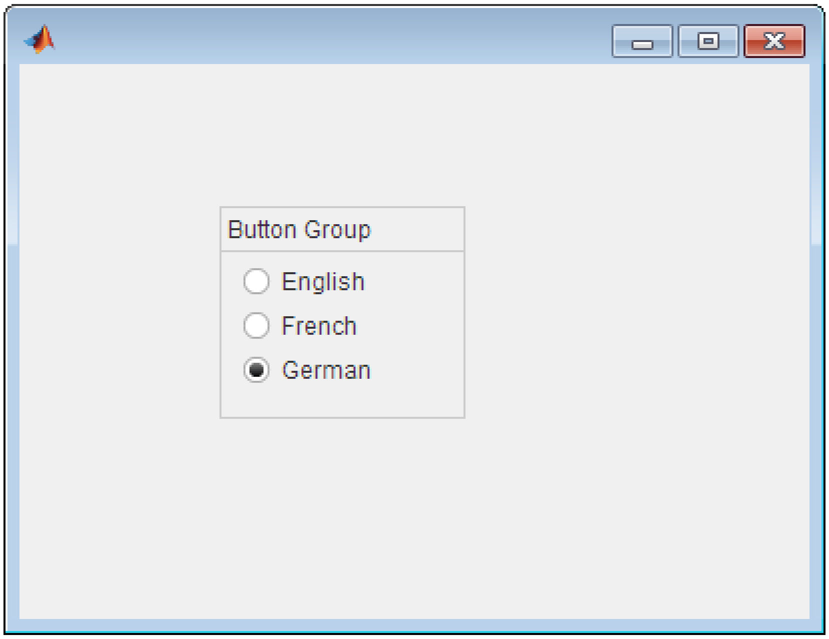
Change the text associated with each radio button.

```
rb1.Text = 'English';
rb2.Text = 'French';
rb3.Text = 'German';
```



Change the radio button selection to German.

```
rb3.Value = true;
```



Determine the font name of the German radio button text.

```
font = rb3.FontName
```

```
font =

Helvetica
```

See Also

`uitogglebutton`

uislider

Create slider component

Syntax

```
slider = uislider
slider = uislider(parent)
slider = uislider(___,Name,Value)
```

Description

`slider = uislider` creates a slider in a new app window.

`slider = uislider(parent)` specifies the object in which to create the slider.

`slider = uislider(___,Name,Value)` specifies slider properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`parent` — Slider parent container
app window object (default) | panel object | button group object

Slider parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

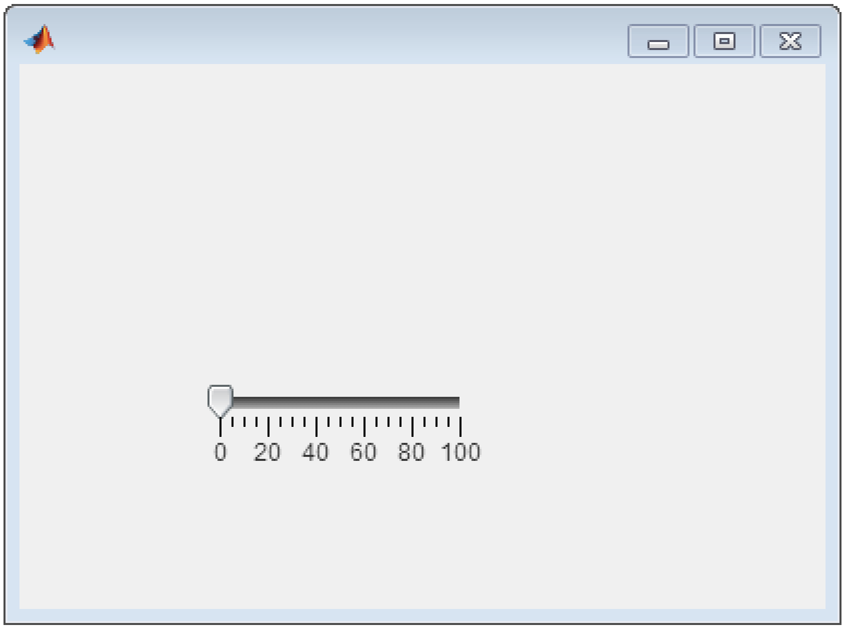
For a list and description of properties (name-value pairs), see Slider Properties.

Example: `'Limits',[0,50]` specifies the minimum slider value as `0`and the maximum slider value as `50`.

Examples

Create a Slider

```
slider = uislider;
```



Specify the Parent Object for a Slider

```
window = appwindow;
slider = uislider(window);
```

Access Slider Property Values

Create a default slider.

```
slider = uislider;
```

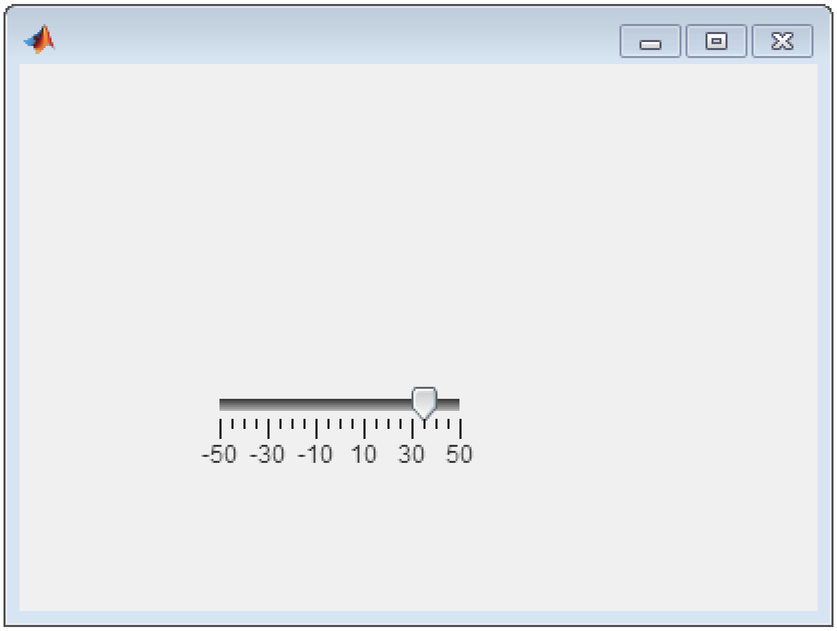
Determine the current slider limits.

```
limits = slider.Limits
```

```
limits =
    0     100
```

Change the slider limits and set the value to 35.

```
slider.Limits = [-50,50];
slider.Value = 35;
```



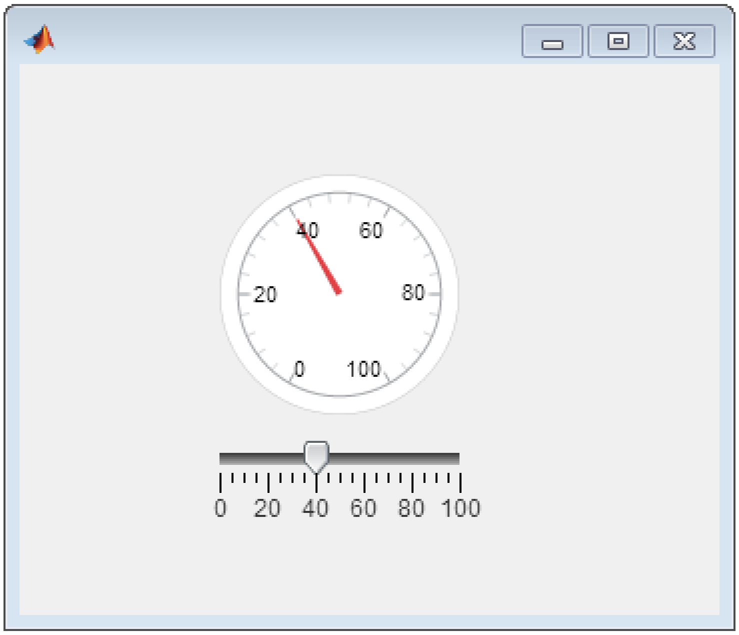
Code the ValueChangedFcn Callback for a Slider

Code the ValueChangedFcn callback for a slider to control the circular gauge needle.

Save the following code to a file on your MATLAB path.

```
function sliderValue
% Create app window and components

window = appwindow('Position',[100,100,350,275]);

gauge = uigauge(window,'Location',[100,100]);

slider = uislider(window,...
    'Location',[100, 75],...
    'ValueChangedFcn',@(slider,event) sliderMoved(slider,gauge));

end

% Create ValueChangedFcn callback
function sliderMoved(slider,gauge)
gauge.Value = slider.Value;
end
```

Run `sliderValue`, and then move the slider. When you release the mouse button, the circular gauge needle moves to the matching value on the gauge.



Code the ValueChangingFcn Callback for a Slider

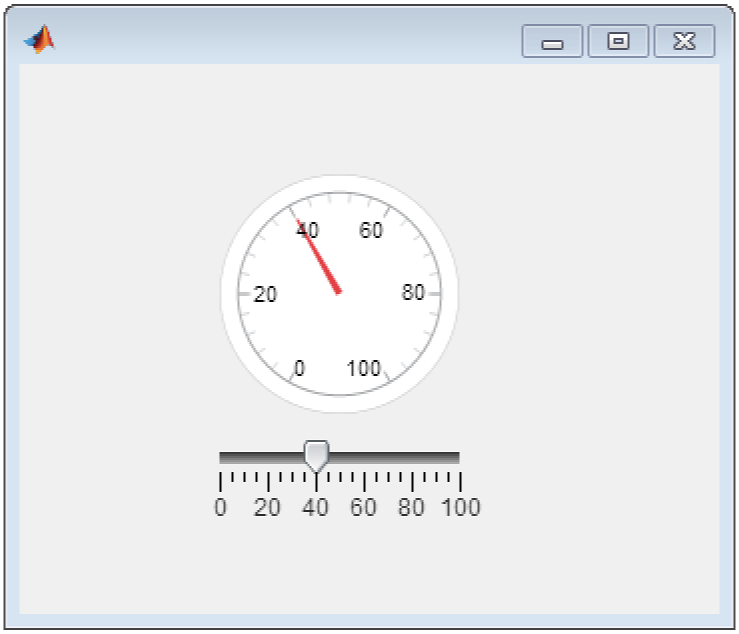
Code the ValueChangingFcn callback for a slider to control the circular gauge needle.

Save the following code to a file on your MATLAB path.

```
function sliderChanging
% Create app window and components

window = appwindow('Position',[100,100,350,275]);
```

```
gauge = uigauge(window,'Location',[100,100]);

slider = uislider(window,...
    'Location',[100, 75],...
    'ValueChangingFcn',@(slider,event) sliderMoving(event,gauge));

end

% Create ValueChangingFcn callback
function sliderMoving(event,gauge)
gauge.Value = event.Value;
end
```

Run `sliderChanging`, and then move the slider. As you move the slider, the circular gauge needle moves, reflecting the slider value.



See Also

Properties
Slider Properties

uiswitch

Create slider switch, rocker switch, or toggle switch component

Syntax

```
control = uiswitch
control = uiswitch(style)
control = uiswitch(parent)
control = uiswitch(parent,style)
control = uiswitch( ___ ,Name,Value)
```

Description

`control = uiswitch` creates a slider switch component in a new app window.

`control = uiswitch(style)` creates a switch of the specified style.

`control = uiswitch(parent)` specifies the object in which to create the switch.

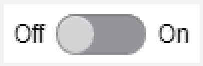
`control = uiswitch(parent,style)` creates a switch of the specified style in the specified parent object.

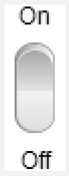
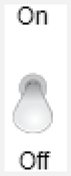
`control = uiswitch( ___ ,Name,Value)` specifies switch properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`style` — Type of switch
`'slider'` (default) | `'rocker'` | `'toggle'`

Type of switch, specified as one of the following styles:

Style	Default Visual
`'slider'`	

Style	Default Visual
`'rocker'`	
`'toggle'`	

`parent` — Switch parent container
app window object (default) | panel object | button group object

Switch parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

For a style-specific list and description of properties, see Switch Properties, Rocker Switch Properties, and Toggle Switch Properties.

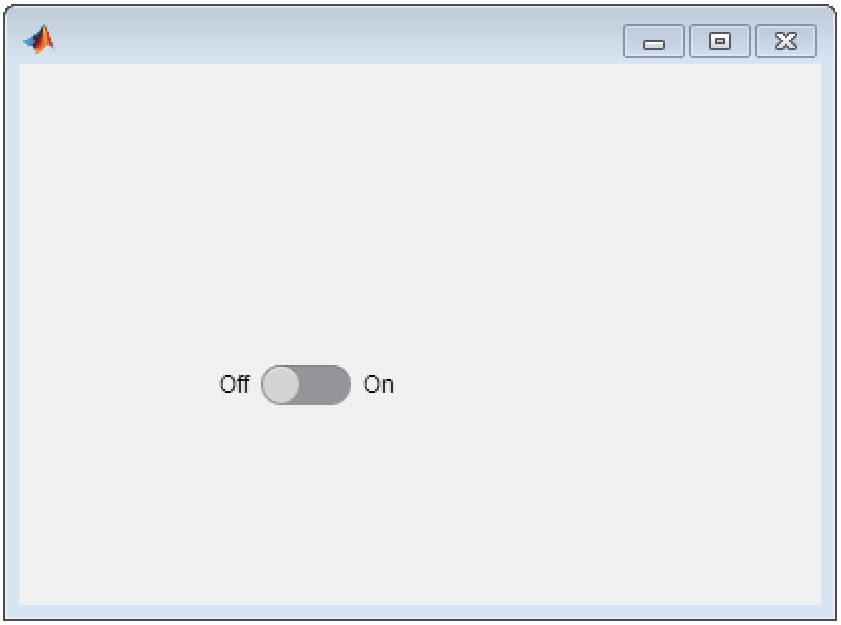
Example: `'Text',{'1','2'} specifies the text that diplays for the two switch states.`

Examples

Create a Slider Switch

Create a slider switch, the default style for a switch.

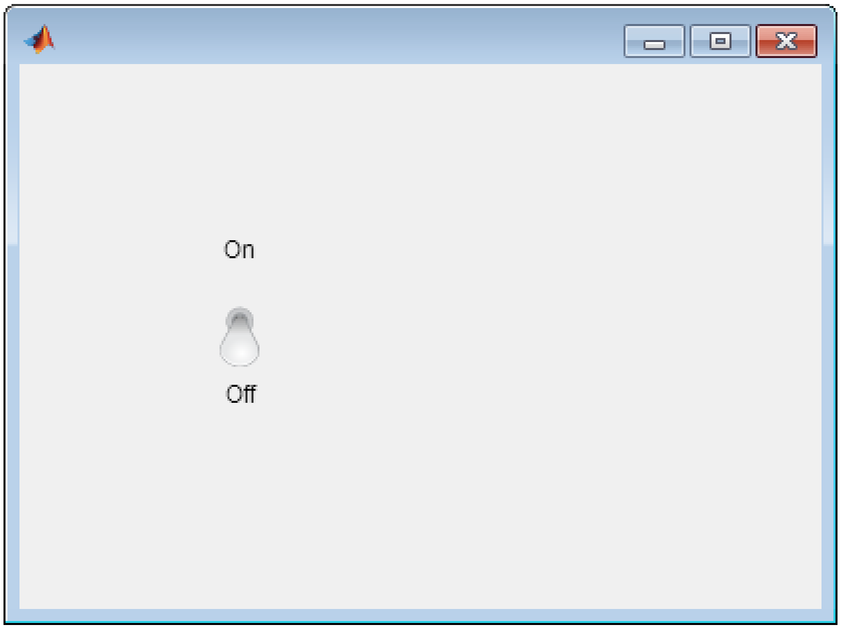
```
sliderswitch = uiswitch;
```



Create a Toggle Switch

Create a toggle switch by specifying the style as toggle.

```
toggleswitch = uiswitch('toggle');
```



Specify the Parent Object for a Rocker Switch

Specify an app window as the parent object for a rocker switch.

```
window = appwindow;
rockerswitch = uiswitch(window,'rocker');
```

Access Switch Property Values

Create a rocker switch.

```
rockerswitch = uiswitch('rocker');
```

Change the switch text.

```
rockerswitch.Text = {'Stop','Start'};
```

Determine the current switch value.

```
val = rockerswitch.Value
```

```
val =
    0
```

MATLAB returns zero, indicating that the switch points downward to the Stop position. (If you change the Orientation property value to `'horizontal'`, the switch points to the left when the `Value` property is zero.)

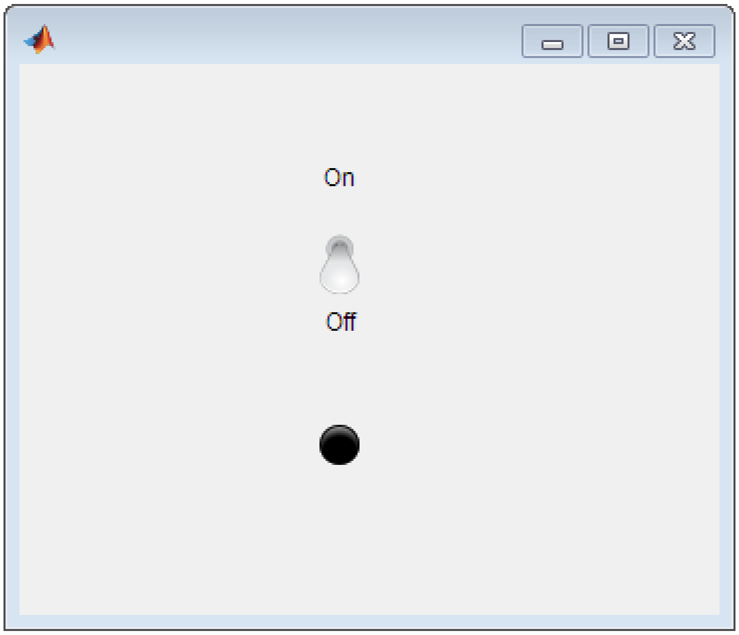
Code the ValueChangedFcn Callback for a Rocker Switch

Create an app window containing a lamp and a rocker switch. Code the callback for the rocker switch so that when a user presses moves the switch, the lamp changes color.

Save the following code to a file on your MATLAB path.

```
function lampSwitch
% Create app window and components

window = appwindow('Position',[100,100,350,275]);

lamp = uilamp(window,...
    'Location',[150,75]);

rockerswitch = uiswitch(window,'toggle',...
    'Location',[150, 160],...
    'Value', true,...
    'ValueChangedFcn',@(rockerswitch,event) switchMoved(rockerswitch,lamp));

end

% Create ValueChangedFcn callback
function switchMoved(rockerswitch,lamp)
switch rockerswitch.Value
    case 1
        lamp.Color = 'green';
    case 0
        lamp.Color = 'black';
end
end
```

Run `lampSwitch`, and then move the switch up and down.



See Also

Properties
Rocker Switch Properties | Switch Properties | Toggle Switch Properties

uitextarea

Create text area component

Syntax

```
textarea = uitextarea
textarea = uitextarea(parent)
textarea = uitextarea(___,Name,Value)
```

Description

`textarea = uitextarea` creates a text area in a new app window.

`textarea = uitextarea(parent)` specifies the object in which to create the text area.

`textarea = uitextarea(___,Name,Value)` specifies text area properties using one or more `Name,Value` pair arguments. Use this option with any of the input argument combinations in the previous syntaxes.

Input Arguments

`parent` — Text area parent container
app window object (default) | panel object | button group object

Text area parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

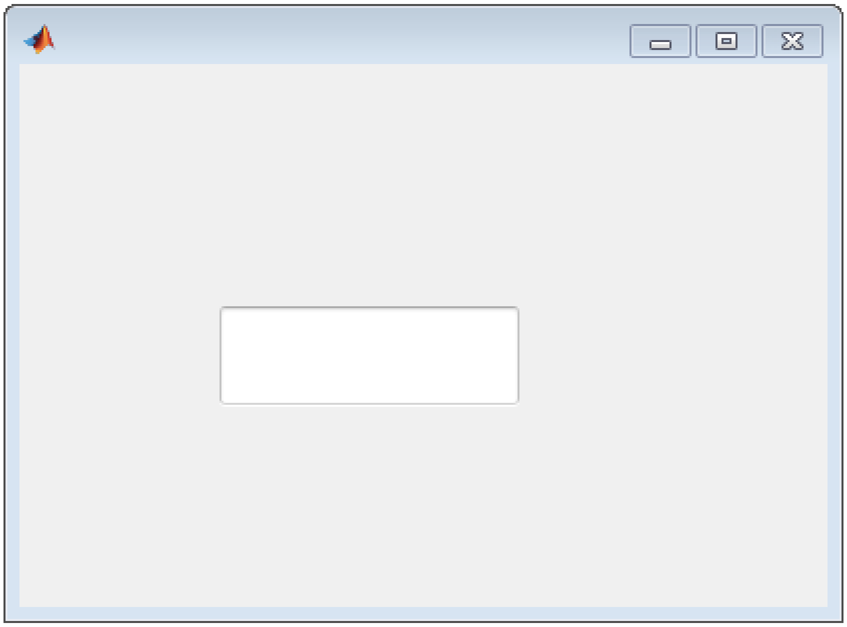
For a list and description of properties, see Text Area Properties.

Example: `'Editable',false` specifies that the user cannot change the text area text.

Examples

Create Text Area

```
textarea = uitextarea;
```



Specify the Parent Object for a Text Area

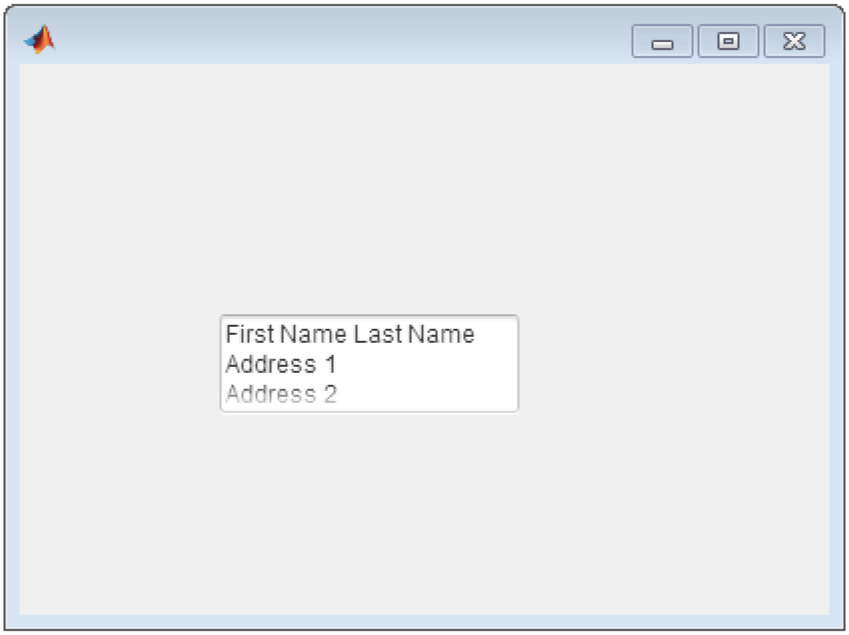
Specify an app window as the parent object for a text area.

```
window = appwindow;
textarea = uitextarea(window);
```

Access Text Area Properties

Create a populated text area.

```
textareafield = uitextarea(...
                'Value', {'First Name Last Name';...
                'Address 1'; 'Address 2';'City, State, Postal Code'});
```


First Name Last Name
Address 1
Address 2

Notice that the default size of the text area is not large enough to display all the specified text.

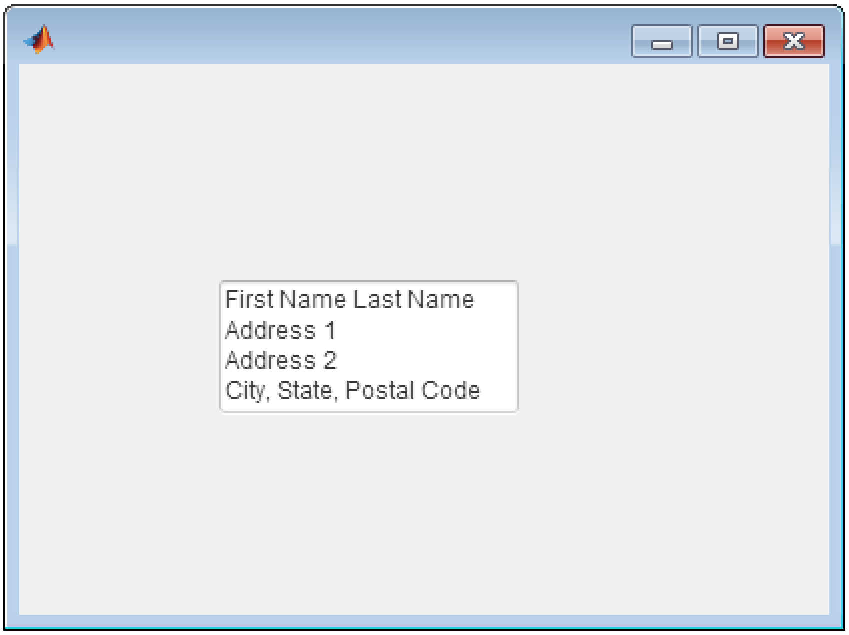
Determine the current size of the text area.

```
size = textareafield.Size
```

```
size =

   150    50
```

Increase the text area size to accommodate all the text.

```
textareafield.Size = [150,67];
```


First Name Last Name
Address 1
Address 2
City, State, Postal Code

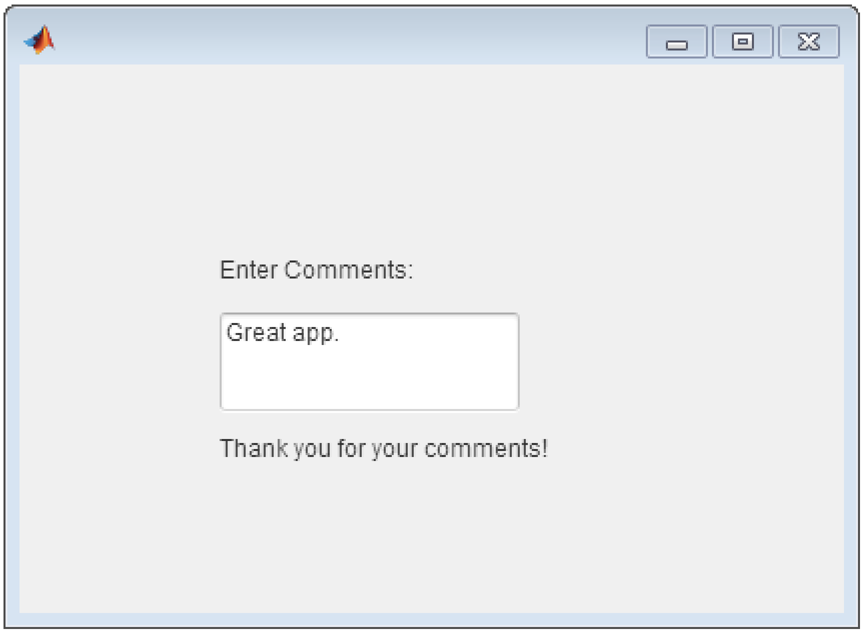
Code the ValueChangedFcn for a Text Area

Create an app window containing a text area and two labels. Code the callback for the text area so that when a user types text and clicks away from the text area, a label is updated to thank the user for the input. If the user removes the text, the thank you text is removed.

```
function comments
% Create app window and components

window = appwindow;

label1 = uilabel(window,...
    'Size',[100,15],...
    'Location',[100,164],...
    'Text', 'Enter Comments:');

label2 = uilabel(window,...
    'Size',[175,15],...
    'Location',[100,75],...
    'Text', '');
```

```
textarea = uitextarea(window,...
    'Location', [100,100],...
    'ValueChangedFcn',@(textarea,event) textEntered(textarea, label2));

% Create ValueChangedFcn callback
    function textEntered(textarea,label2)
        val = textarea.Value;
        label2.Text = '';
        % Check each element of text area cell array for text
        for k = 1:length(val)
            if(~isempty(val{k}))
                label2.Text = 'Thank you for your comments!';
                break;
            end
        end
    end
end
```

Run `comments`, and type text in the text area field. Click away from the text area to trigger the callback.



See Also

Functions
`uieditfield`

Properties
Text Area Properties

uitogglebutton

Create toggle button component

Syntax

```
togglebutton = uitogglebutton(parent)
togglebutton = uitogglebutton(parent,Name,Value)
```

Description

`togglebutton = uitogglebutton(parent)` creates a toggle button within the specified parent button group.

`togglebutton = uitogglebutton(parent,Name,Value)` specifies toggle button properties using one or more `Name,Value` pair arguments.

Input Arguments

`parent` — Toggle button parent container
app window object (default) | panel object | button group object

Toggle button parent container, specified as an app window, panel, or button group object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Text', 'French'` specifies that the text, `French`, displays on the toggle button.

Examples

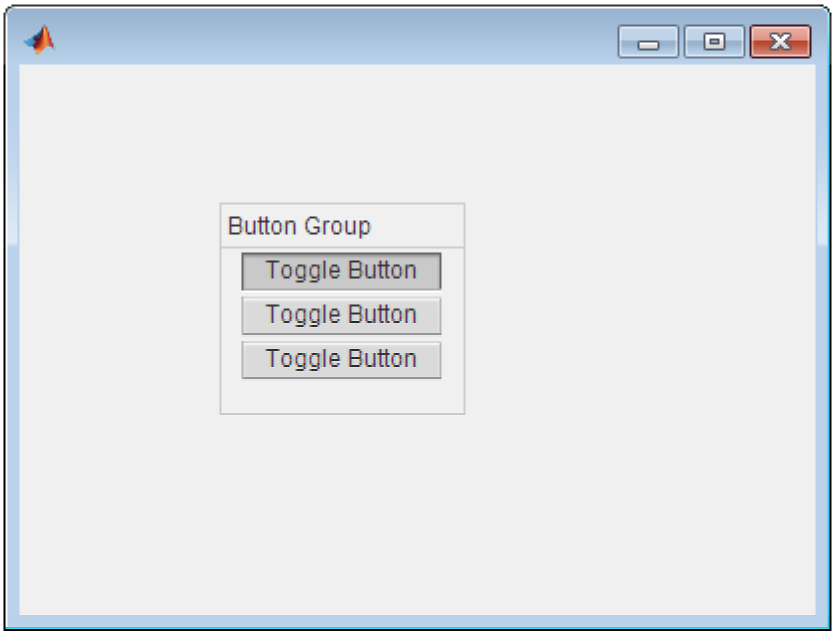
Create Toggle Buttons Within a Button Group and Access Property Values

To create a toggle button, first create an app window and a button group object.

```
window = appwindow;
bg = matlab.ui.control.ButtonGroup('Parent',window);
```

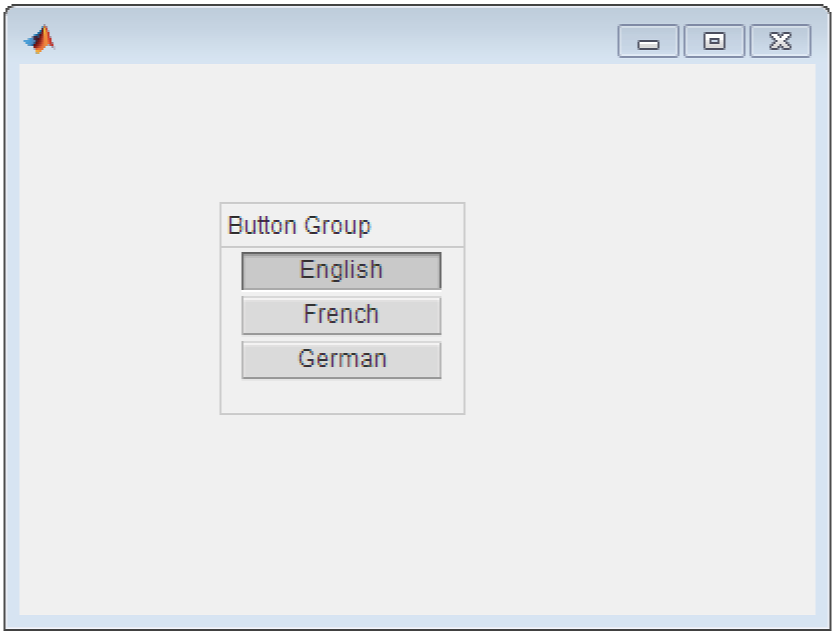
Create three toggle buttons and specify the location of each.

```
tb1 = uitogglebutton(bg,'Location',[10,60]);
tb2 = uitogglebutton(bg,'Location',[10,38]);
tb3 = uitogglebutton(bg,'Location',[10,16]);
```



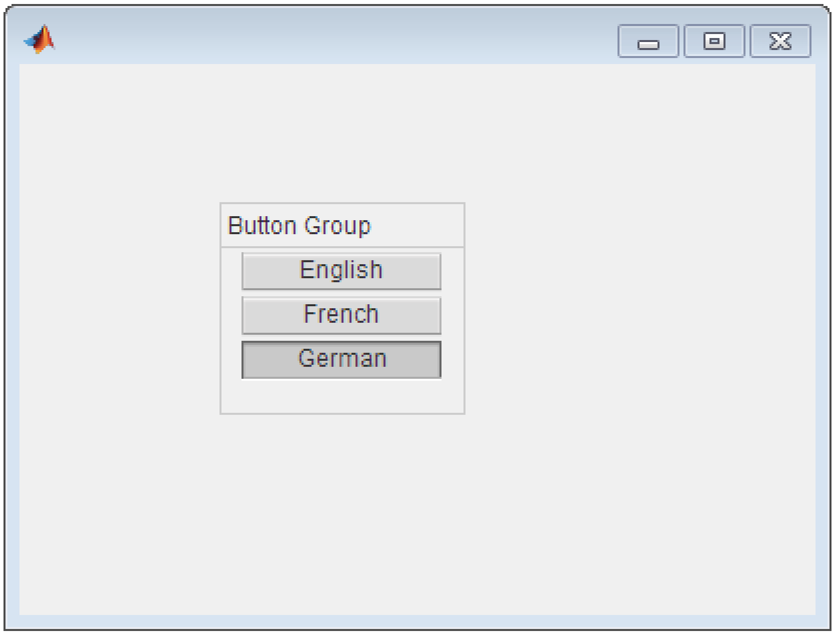
Change the text associated with each toggle button.

```
tb1.Text = 'English';
tb2.Text = 'French';
tb3.Text = 'German';
```



Change the toggle button selection to German.

```
tb3.Value = true;
```



Determine the font name of the German toggle button text.

```
font = tb3.FontName
```

```
font =

Helvetica
```

See Also

`uiradiobutton`