L37: Graphical User Interface Tools

An introduction

Lucas A. J. Bastien

E7 Spring 2017, University of California at Berkeley

April 24, 2017

Version: release

Announcements

Project is due on Friday April 28 at 11:59 pm (midnight)

- **Your code MUST work on the computers in 1109 Etcheverry Hall**
  (It is where we will do the grading)
  - Do not use functions from Toolboxes not installed on these computers
  - Your personal computer may be faster than these computers
  - Some "newer syntax" may not be available on these computers
- **It is your responsibility to make sure that your code works on these computers**
  - **We will NOT debug your code before grading it**
- There can be transparent ghosts

Today:

- Graphical User Interface Tools

Wednesday:

- Guest lecture

Friday:

- Practice questions

Variable number of input arguments

One can define user-defined functions that take an arbitrary number of input arguments by using "`varargin`" as the last input argument

Inside the function:

- `varargin`: `cell` array of all the remaining inputs
- `nargin`: number of input arguments

Example:

```
function [y] = my_varargin_example(x, varargin)

fprintf('There are %d input argument(s).\n', nargin)
for i = 1:nargin-1
    fprintf('Argument number %d is of class %s.\n', ...
        i+1, class(varargin{i}))
end

y = x + 2;

end
```

Variable number of input arguments

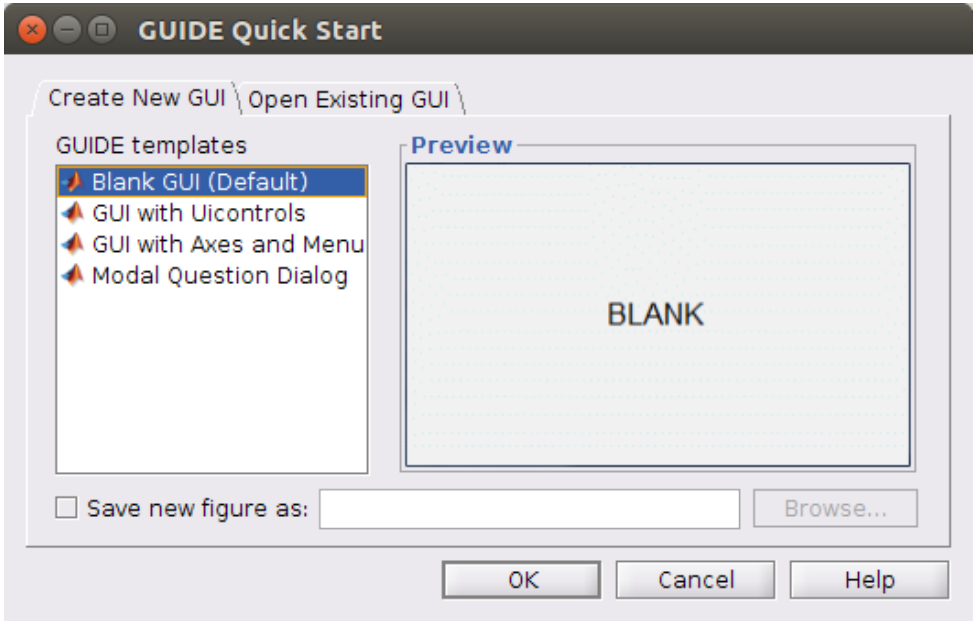
```
>> my_varargin_example(3)
There are 1 input argument(s).
ans =
     5

>> my_varargin_example(3, 'hello', 10)
There are 3 input argument(s).
Argument number 2 is of class char.
Argument number 3 is of class double.
ans =
     5

>> my_varargin_example(3, 'hello', {4}, [3, 5], NaN)
There are 5 input argument(s).
Argument number 2 is of class char.
Argument number 3 is of class cell.
Argument number 4 is of class double.
Argument number 5 is of class double.
ans =
     5
```

GUIDE: Graphical user interface design environment

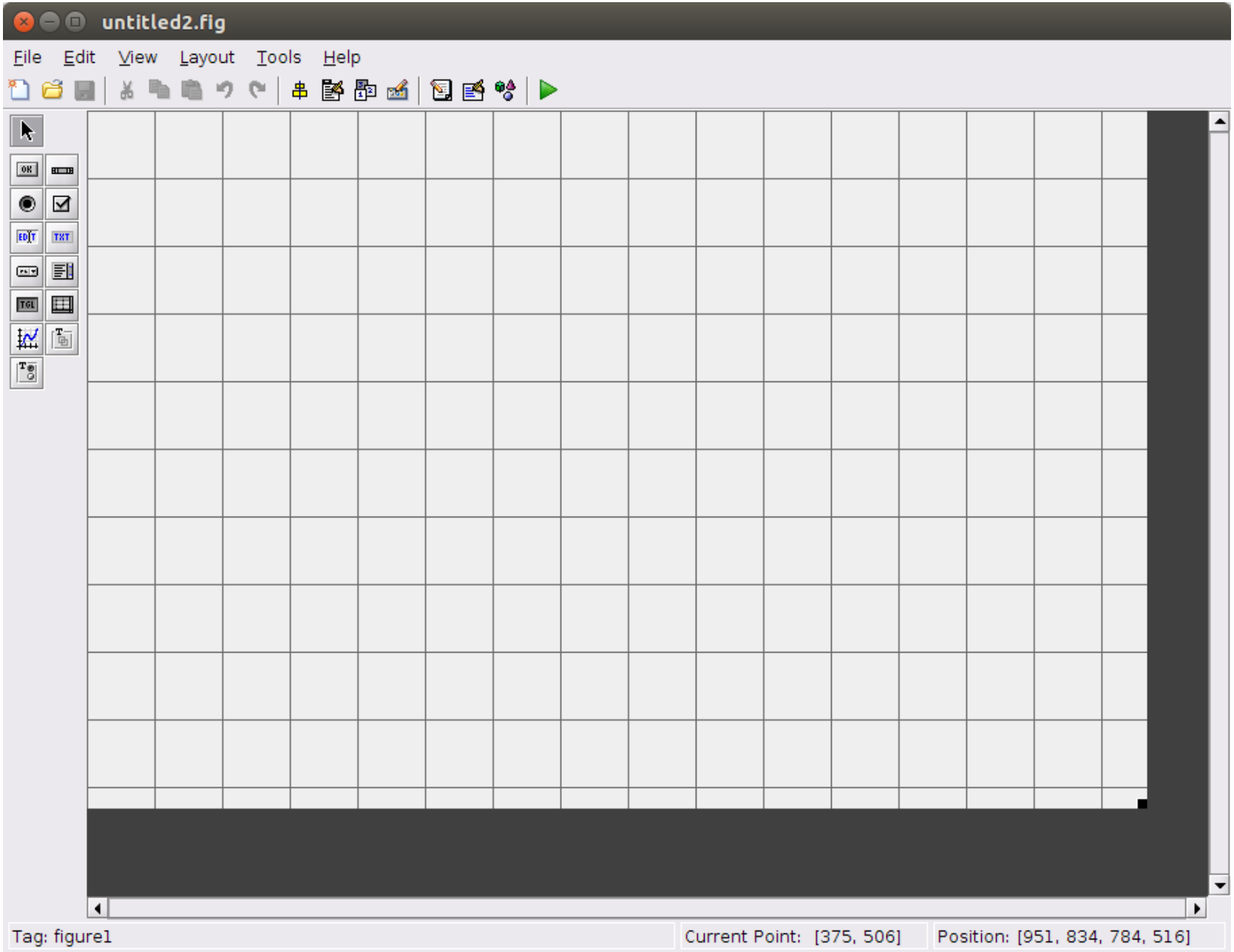
GUI: Graphical User Interface

GUIDE: a Matlab user interface to create Matlab user interfaces!

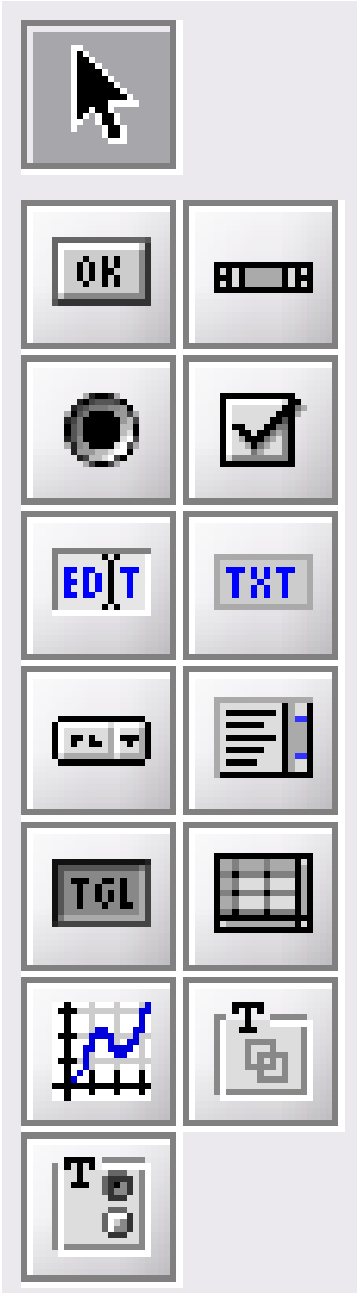
```
>> % Open the GUIDE "quick start" window
>> % to create a new GUI
>> guide

>> % Edit an existing GUI (".fig" file)
>> guide(filename)
```



Start with an empty GUI



Add elements to the GUI



Push button

Slide bar

Radio button

Check box

Editable text box

Static text box

Pop-up menu

List box

Toggle button

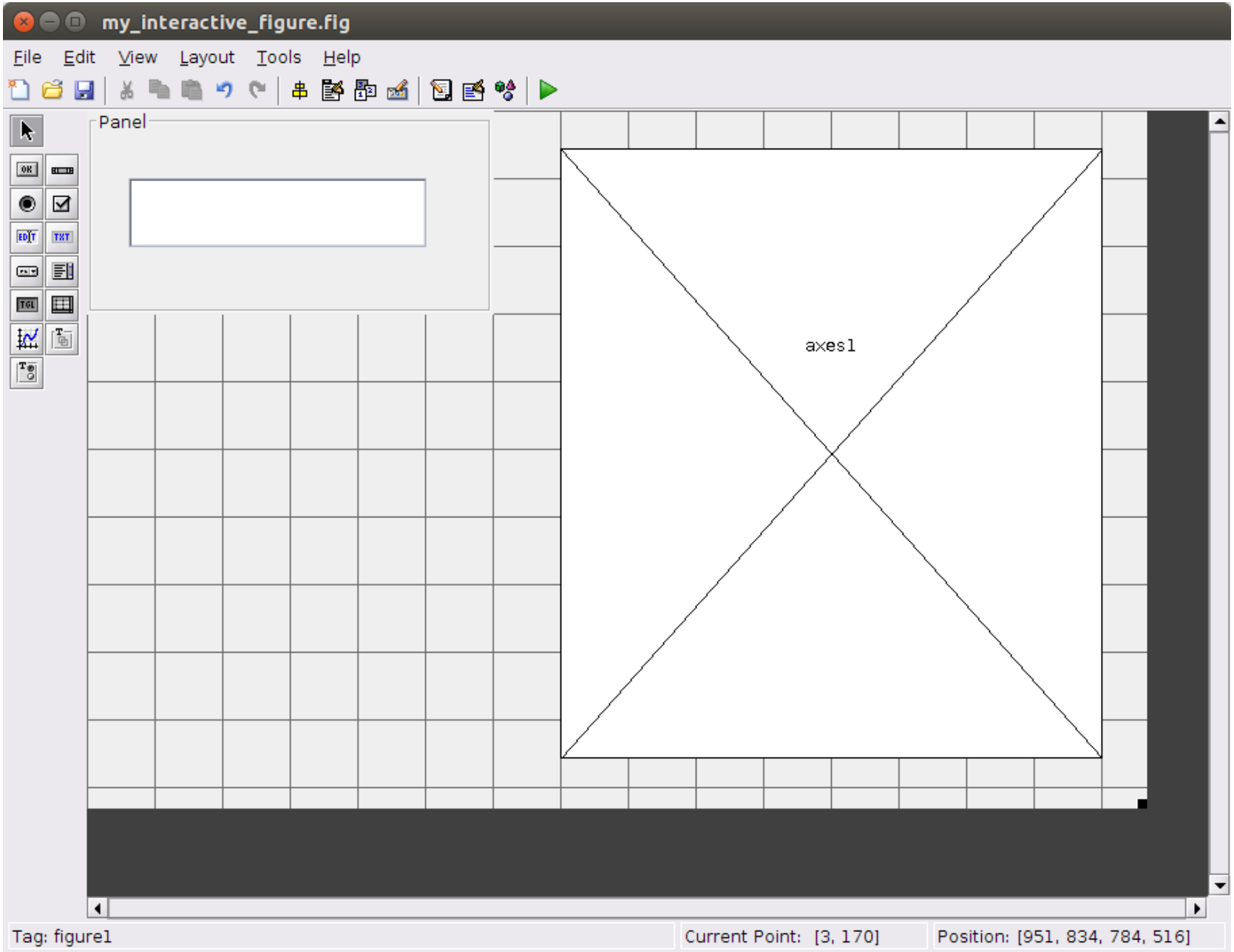
Table

Axes

Panel with title

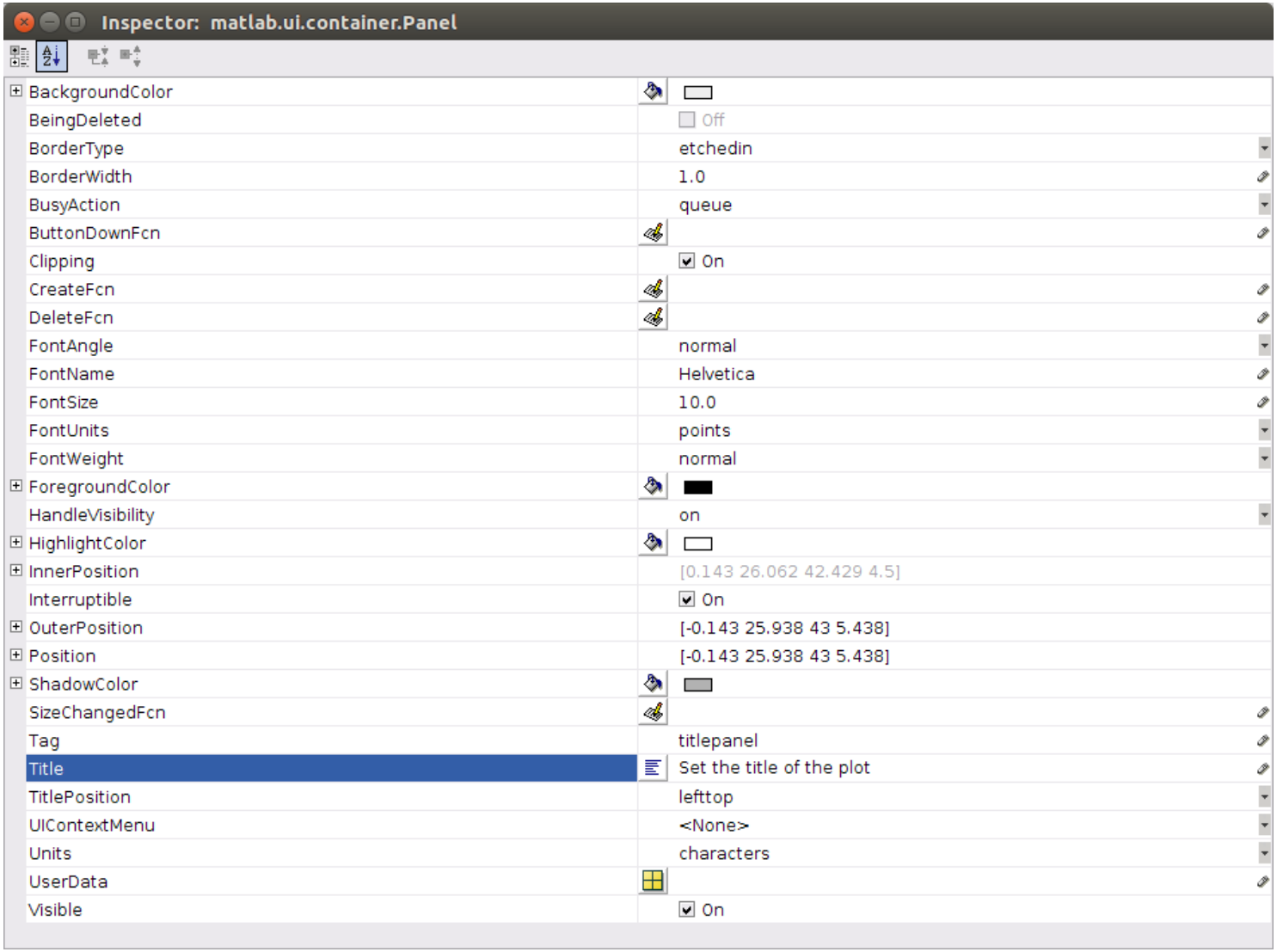
Button group

Add axes, a panel, and an editable text box

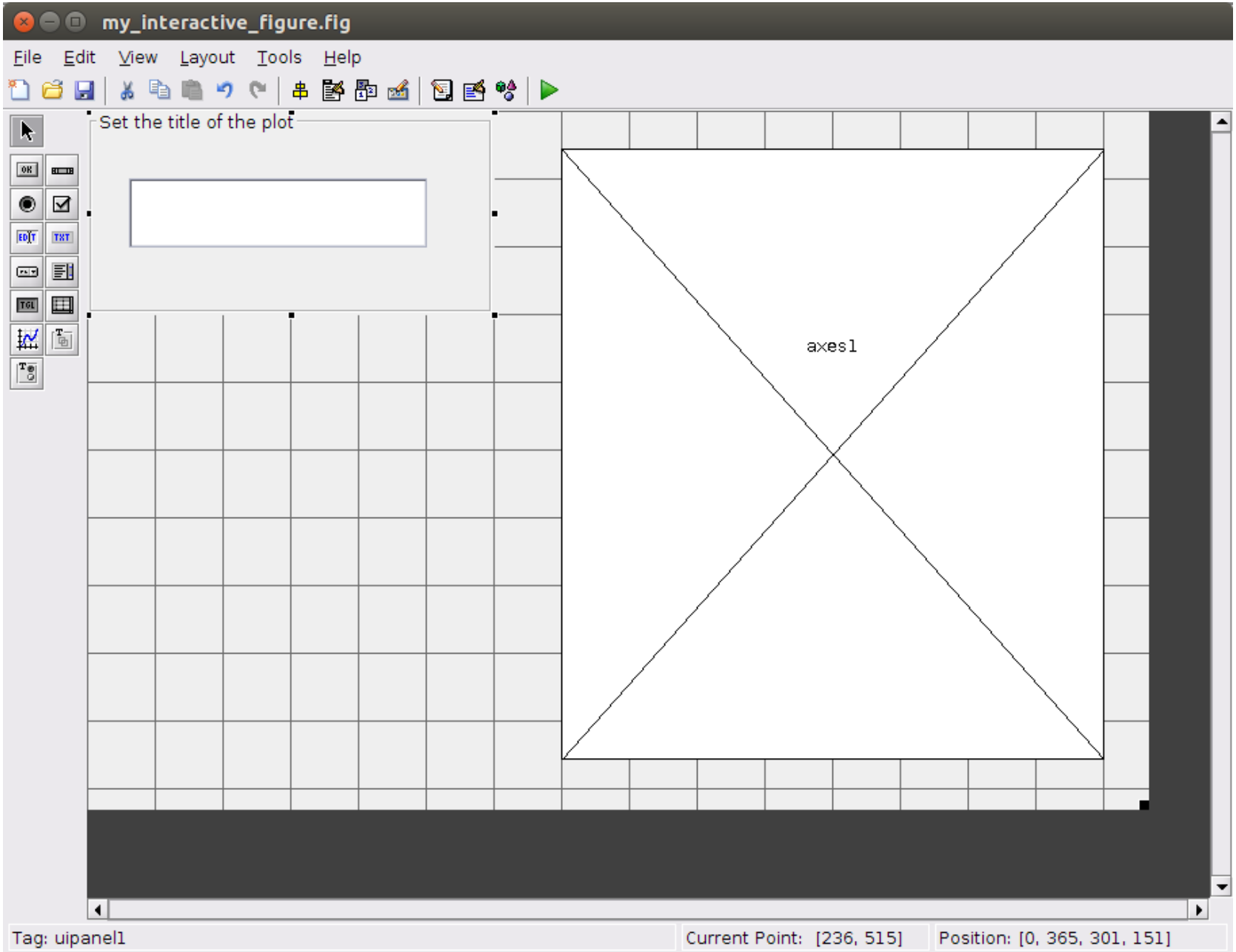


Edit the properties of the panel

Double-click on the panel, and change the "Tag" (*i.e.* the "nickname" of the object) and the "Title" (*i.e.* the default text)



Edit the properties of the panel



Callback functions

In this context, **callback functions** are functions that are executed when certain events happen *e.g.*, when a push button is pushed

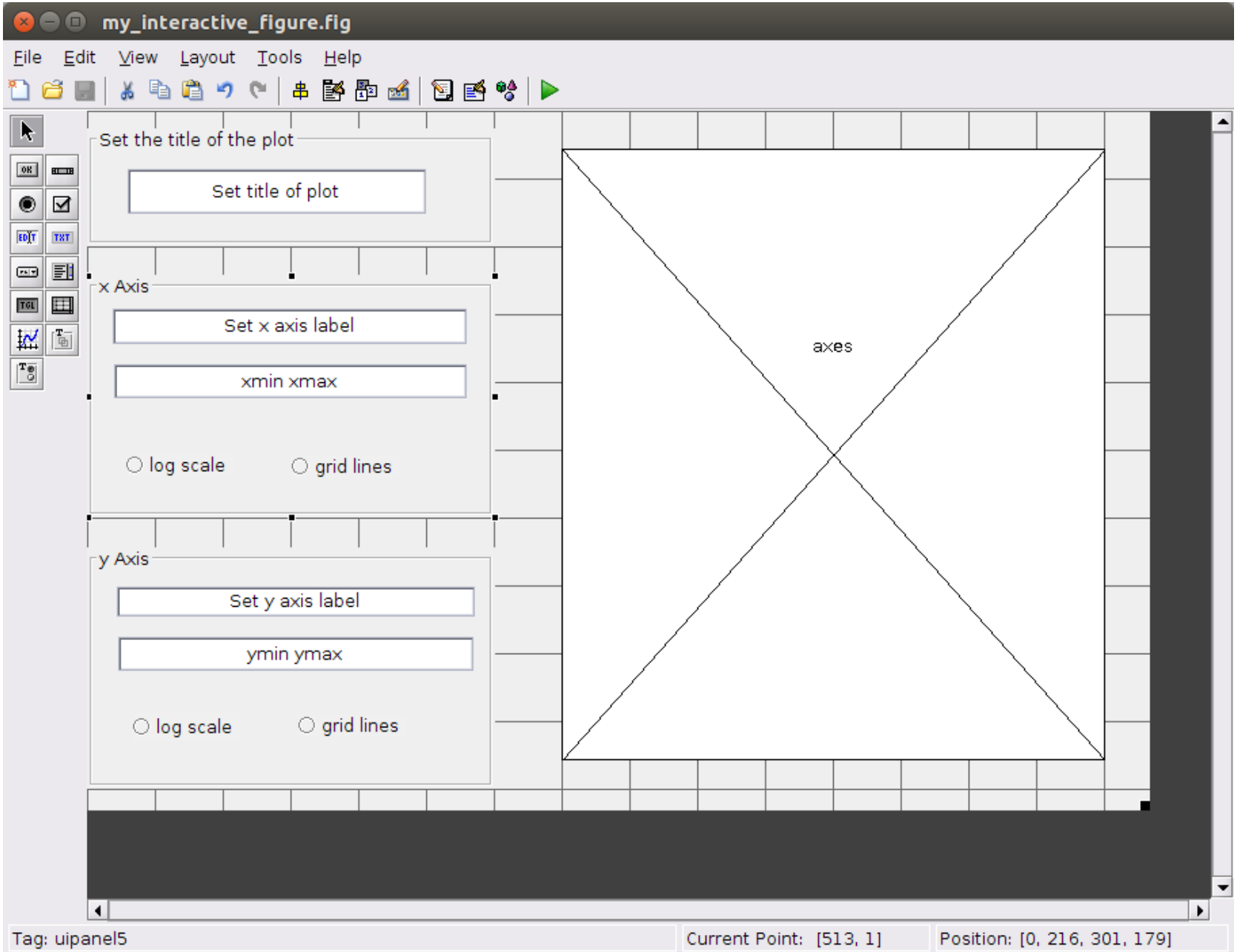
Example: add a callback function associated to the editable text box that sets the title of the plot to the content of the text box

```
77    function plottitle_Callback(hObject, eventdata, handles)
78    % hObject    handle to plottitle (see GCBO)
79    % eventdata  reserved - to be defined in a future version of MATLAB
80    % handles    structure with handles and user data (see GUIDATA)
81
82    % Hints: get(hObject,'String') returns contents of plottitle as text
83    %        str2double(get(hObject,'String')) returns contents of plottitle as a double
84 -  txt = get(hObject,'String');
85 -  title(txt)
```

The name of the callback function depends on the object's tag:

String	Set title of plot
Style	edit
Tag	plottitle
TooltipString	
UIContextMenu	<None>

Add other elements to control the plot



Callback functions

Setting the label of the x axis:

```
function xlabel_Callback(hObject, eventdata, handles)
% hObject    handle to xlabel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xlabel as text
%        str2double(get(hObject,'String')) returns contents of xlabel as a double
txt = get(hObject,'String');
xlabel(txt)
```

Setting the limits of the x axis:

```
function xlimits_Callback(hObject, eventdata, handles)
% hObject    handle to xlimits (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xlimits as text
%        str2double(get(hObject,'String')) returns contents of xlimits as a double
txt = get(hObject,'String');
xlimits = str2num(txt);
xlim(xlimits)
```

Callback functions

Toggle log-scale on the x axis:

```
% --- Executes on button press in xlogscale.
function xlogscale_Callback(hObject, eventdata, handles)
% hObject    handle to xlogscale (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of xlogscale
toggled = get(hObject,'Value');
if toggled
    set(gca(), 'XScale', 'log');
else
    set(gca(), 'XScale', 'linear');
end
```

Callback functions

Toggle grid lines on the x axis:

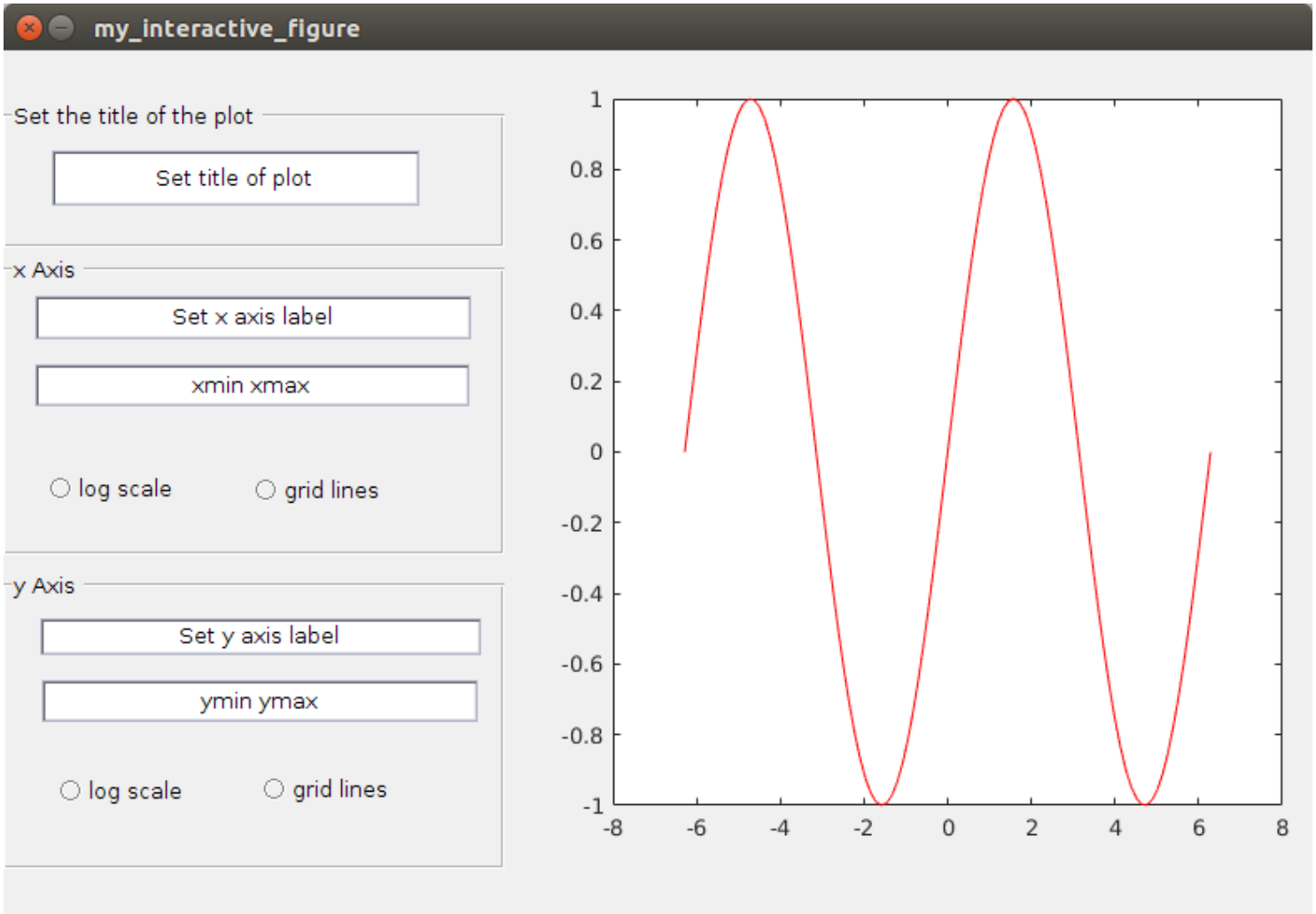
```
% --- Executes on button press in xgrid.
function xgrid_Callback(hObject, eventdata, handles)
% hObject    handle to xgrid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of xgrid
toggled = get(hObject,'Value');
if toggled
    set(gca(), 'XGrid', 'on');
else
    set(gca(), 'XGrid', 'off');
end
```

Have the GUI plot the data passed as input arguments

```
% --- Executes just before my_interactive_figure is made visible.
function my_interactive_figure_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to my_interactive_figure (see VARARGIN)

% Choose default command line output for my_interactive_figure
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes my_interactive_figure wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% Plot data given in input arguments
if nargin < 6
    plot_style = 'r-';
else
    plot_style = varargin{3}
end
plot(varargin{1}, varargin{2}, plot_style)
```

Test our new graphical user interface

```
>> x = linspace(-2*pi, 2*pi, 100);
>> y = sin(x);
>> my_interactive_figure(x, y)
```



Adding user interface objects using uicontrol

Use `uicontrol` to add user interface objects to an existing figure. Syntax:

```
>> uicontrol(PropertyName1, PropertyValue1, ...
       PropertyName2, PropertyValue2, ...)
```

Example:

```
>> fig = figure();
>> pushbutton = uicontrol('Parent', fig, ...
    'Style', 'pushbutton', 'String', 'Click here!', ...
    'Position', [20, 300, 200, 40], ...
    'Callback', @(o, e, h) fprintf('Hello\n'))
```

Useful properties:

- `'Parent'`: object into which to create the new object
- `'Style'`: type of object to create
- `'Position'`: position and size of the new object
- `'String'`: text displayed in the object
- `'Callback'`: callback function when object is "activated"

See Matlab's documentation for more detailed information

Show all properties of an object (method 1)

```
>> fig = figure();
>> pushbutton = uicontrol('Parent', fig, ...
    'Style', 'pushbutton', 'String', 'Click here!', ...
    'Position', [20, 300, 200, 40], ...
    'Callback', @(o, e, h) fprintf('Hello\n'))
```

Click on "all properties" to show the list of the properties of the object

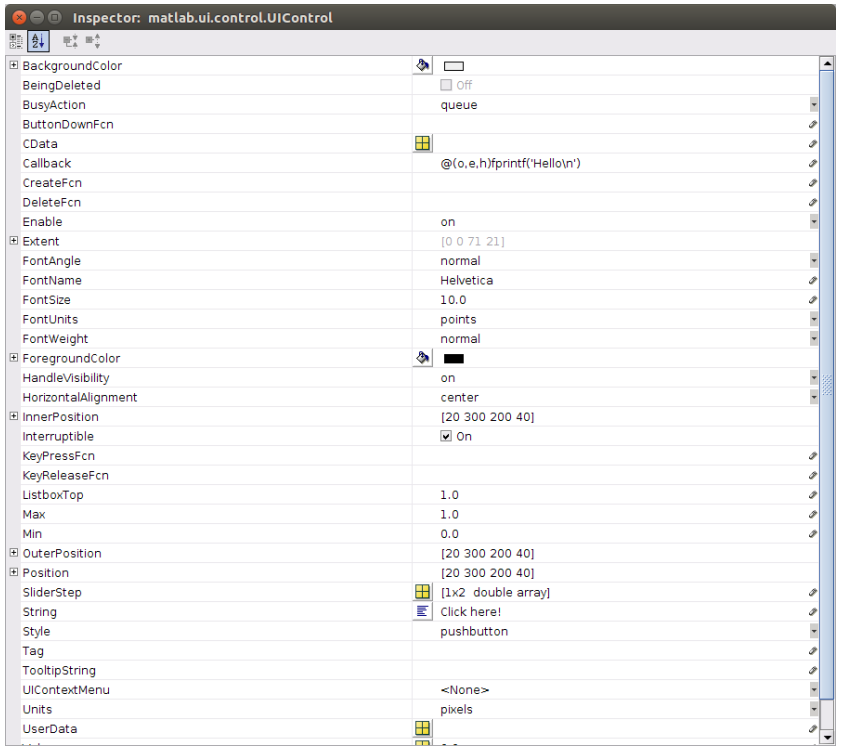
```
>> fig = figure();
pushbutton = uicontrol('Parent', fig, ...
    'Style', 'pushbutton', 'String', 'Click here!', ...
    'Position', [20, 300, 200, 40], ...
    'Callback', @(o, e, h) fprintf('Hello\n'))
pushbutton =
  UIControl (Click here!) with properties:

              Style: 'pushbutton'
             String: 'Click here!'
    BackgroundColor: [0.9400 0.9400 0.9400]
           Callback: @(o,e,h)fprintf('Hello\n')
              Value: 0
           Position: [20 300 200 40]
              Units: 'pixels'

  Show all properties
fx >>
```

```
Show all properties
      BackgroundColor: [0.9400 0.9400 0.9400]
         BeingDeleted: 'off'
           BusyAction: 'queue'
        ButtonDownFcn: ''
                CData: []
             Callback: @(o,e,h)fprintf('Hello\n')
             Children: [0×0 handle]
            CreateFcn: ''
            DeleteFcn: ''
               Enable: 'on'
               Extent: [0 0 71 21]
            FontAngle: 'normal'
             FontName: 'Helvetica'
             FontSize: 10
            FontUnits: 'points'
           FontWeight: 'normal'
      ForegroundColor: [0 0 0]
    HandleVisibility: 'on'
  HorizontalAlignment: 'center'
        InnerPosition: [20 300 200 40]
        Interruptible: 'on'
          KeyPressFcn: ''
        KeyReleaseFcn: ''
           ListboxTop: 1
                  Max: 1
                  Min: 0
        OuterPosition: [20 300 200 40]
               Parent: [1×1 Figure]
             Position: [20 300 200 40]
           SliderStep: [0.0100 0.1000]
               String: 'Click here!'
                Style: 'pushbutton'
                  Tag: ''
        TooltipString: ''
                 Type: 'uicontrol'
        UIContextMenu: [0×0 GraphicsPlaceholder]
                Units: 'pixels'
             UserData: []
                Value: 0
              Visible: 'on'
fx >>
```

Show all properties of an object (method 2)

Use Matlab's builtin function `inspect`

```
>> fig = figure();
>> pushbutton = uicontrol('Parent', fig, ...
    'Style', 'pushbutton', 'String', 'Click here!', ...
    'Position', [20, 300, 200, 40], ...
    'Callback', @(o, e, h) fprintf('Hello\n'));
>> inspect(pushbutton)
```



Adding user interface objects using uicontrol

Other examples:

```
>> fig = figure();

>> on_or_off = {'off', 'on'};
>> cb = @(o, e, h) fprintf('Radio button is %s.\n', ...
          on_or_off{get(o, 'Value')+1})

>> radiobutton = uicontrol('Parent', fig, ...
    'Style', 'radiobutton', 'String', 'my on or off', ...
    'Position', [010, 10, 100, 50], 'Callback', cb)
```

```
>> fig = figure();
>> pushbutton = uicontrol('Parent', fig, ...
    'Style', 'pushbutton', 'String', 'Add rectangle', ...
    'Position', [0, 0, 200, 50], ...
    'Callback', @my_random_rectangle)
```

Other styles: 'edit', 'checkbox', 'togglebutton', ...

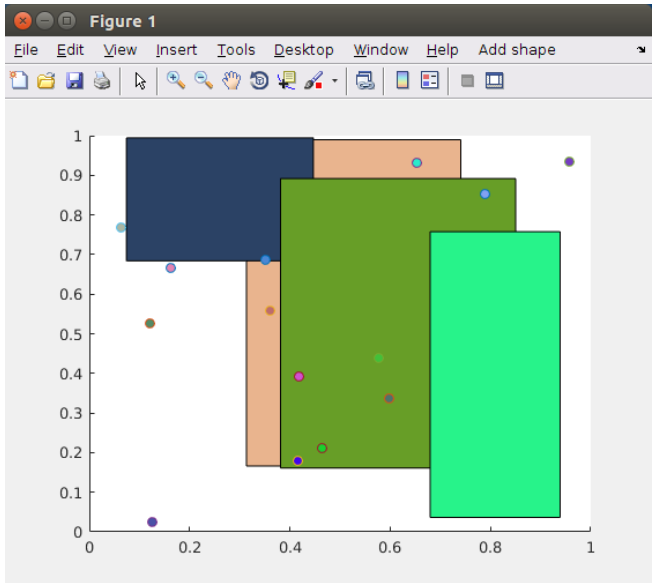
my_random_rectangle

```
function [] = my_random_rectangle(o, e, h)

% Add a pseudo-randomly generated rectangle to the current
% axis object.

x = rand();
y = rand();
width = rand()*(1-x);
height = rand()*(1-y);
color = rand(1, 3);

rectangle('Parent', gca(), ...
    'Position', [x, y, width, height], ...
    'FaceColor', color);

end
```

Adding menu objects using uimenu

Use uimenu to add menu objects to an existing figure. Example:

```
>> fig = figure();
>> hold on
>> menu_shapes = uimenu('Parent', fig, 'Label', 'Add shape')
>> menu_rectangle = uimenu('Parent', menu_shapes, ...
    'Label', 'Add rectangle', 'CallBack', @my_random_rectangle)
>> add_dot = @(o, e, h) plot(rand(), rand(), 'o', ...
    'MarkerFaceColor', rand(1, 3));
>> menu_dot = uimenu('Parent', menu_shapes, ...
    'Label', 'Add dot', 'CallBack', add_dot)
```



More graphical user interface tools

Choose a color:

```
>> my_color = uisetcolor();
```

Choose a font:

```
>> my_font = uisetfont();
>> % Set the font of the current figure to my_font
>> set(gca(), my_font)
```

Choose a file:

```
>> % For a file that already exists
>> my_filename = uigetfile();
>> % For a file that may not exist
>> my_filename = uiputfile();
```