CM2202: Scientific Computing and Multimedia Applications
Lab Class Week 4

School of Computer Science & Informatics

Manually Creating Uicontrol Elements

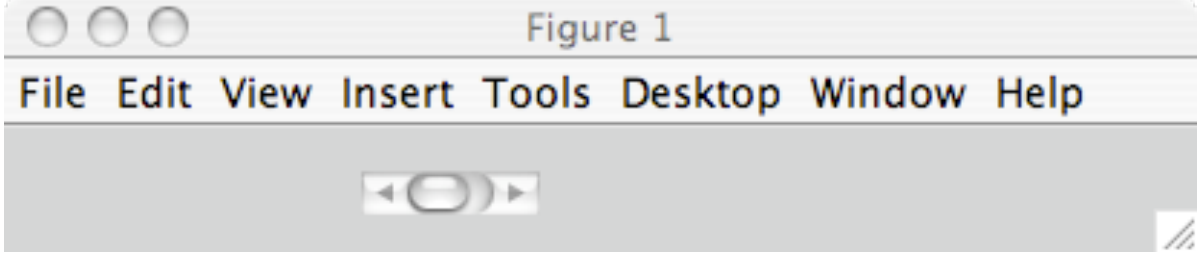
To create a `uicontrol` element, use the MATLAB command:

```
handle = uicontrol('Property1Name', Property1Value, ...
                   Property2Name', Property2Value, ...
                   .
                   .
                   );
```

- The first property name usually sets the style: Check box, slider, *etc.*
- Others specify attributes of that object.
- Simple Example:

```
h_slider = uicontrol('Style','slider',...
'units','normalized',...
'position',[.3 .6 .15 .05]);
```

- Use `doc uicontrol` and links to find detailed Uicontrol Properties



Uicontrol Callbacks

Having created a UI element such as a slider, we need to attach a callback to the element:

- Simply set the `'callback'` property value with an appropriate MATLAB function, e.g.

```
h_slider = uicontrol(h_fig,...
'callback','slidergui(''Slider Moved'');',...
```

- Callback can be a *self-referenced* function (as in example below) or an entirely new function (see GUIDE example later).
- Within the callback, you need to access the value of the Uicontrol element:
  - Store data in graphics handle `'userdata'`:

```
set(h_fig,'userdata', h_slider);
```

  - Retrieve values via a few `get`s:

```
h_slider = get(gcf,'userdata');
value = get(h_slider,'value');
```

Full Slide Callback Code Example

```
function slidergui(command_str)
% Slider
%
% Simple Example of creating slider GUIs.

if nargin < 1
command_str = 'initialize';
end

if strcmp(command_str,'initialize')
        h_fig = figure(1);  clf;

        h_slider = uicontrol(h_fig,...
             'callback','slidergui(''Slider Moved'');',...
             'style','slider',...
             'min',-100,'max',100,...
             'position',[25 20 150 20]);

        set(h_fig,'userdata',h_slider);
else
        h_slider = get(gcf,'userdata');
        value = get(h_slider,'value');
        disp(value);
end;
```

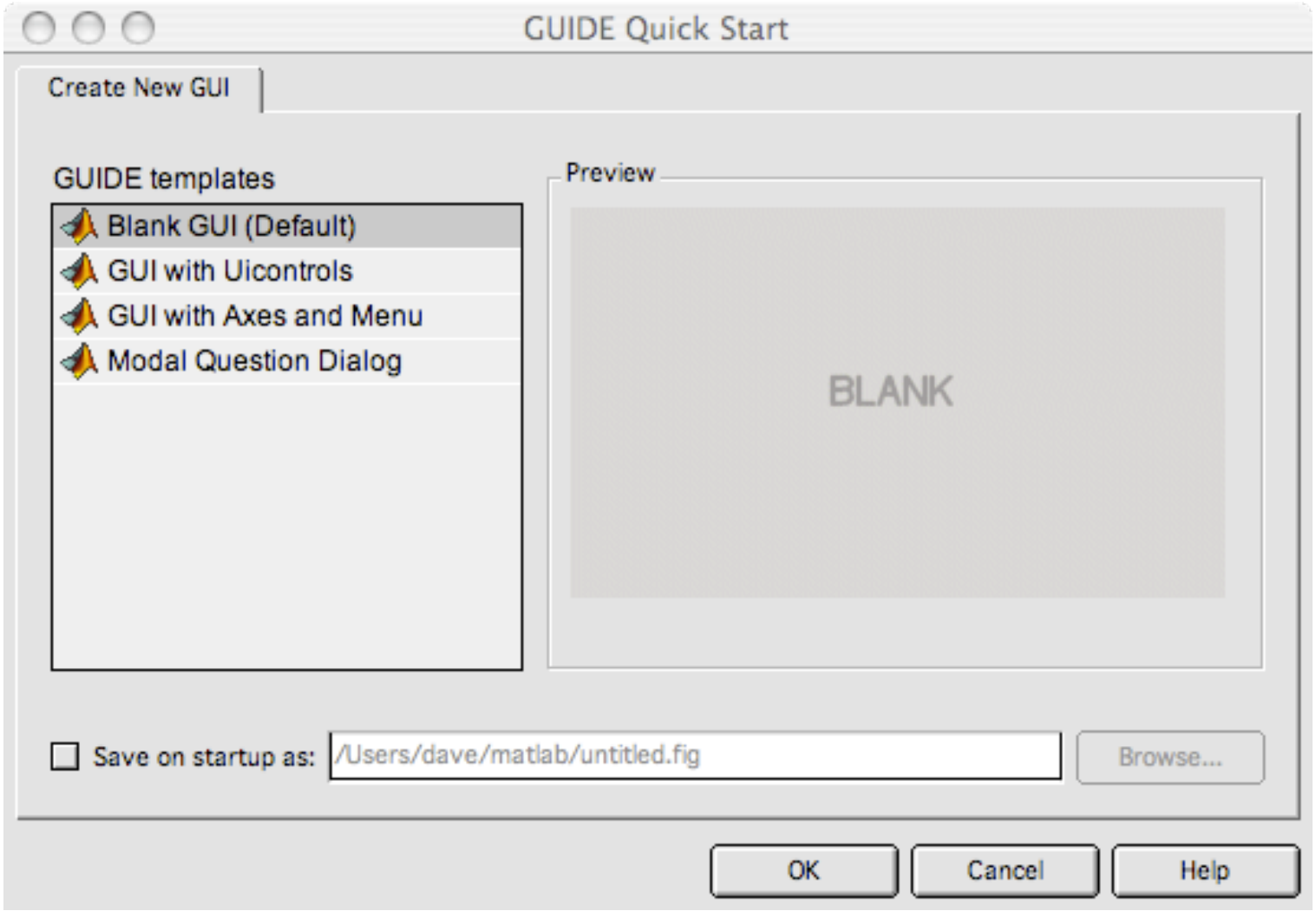
MATLAB's Graphical User Interface Development Environment — GUIDE

GUIDE provides a WYSIWYG way to assemble your GUI:

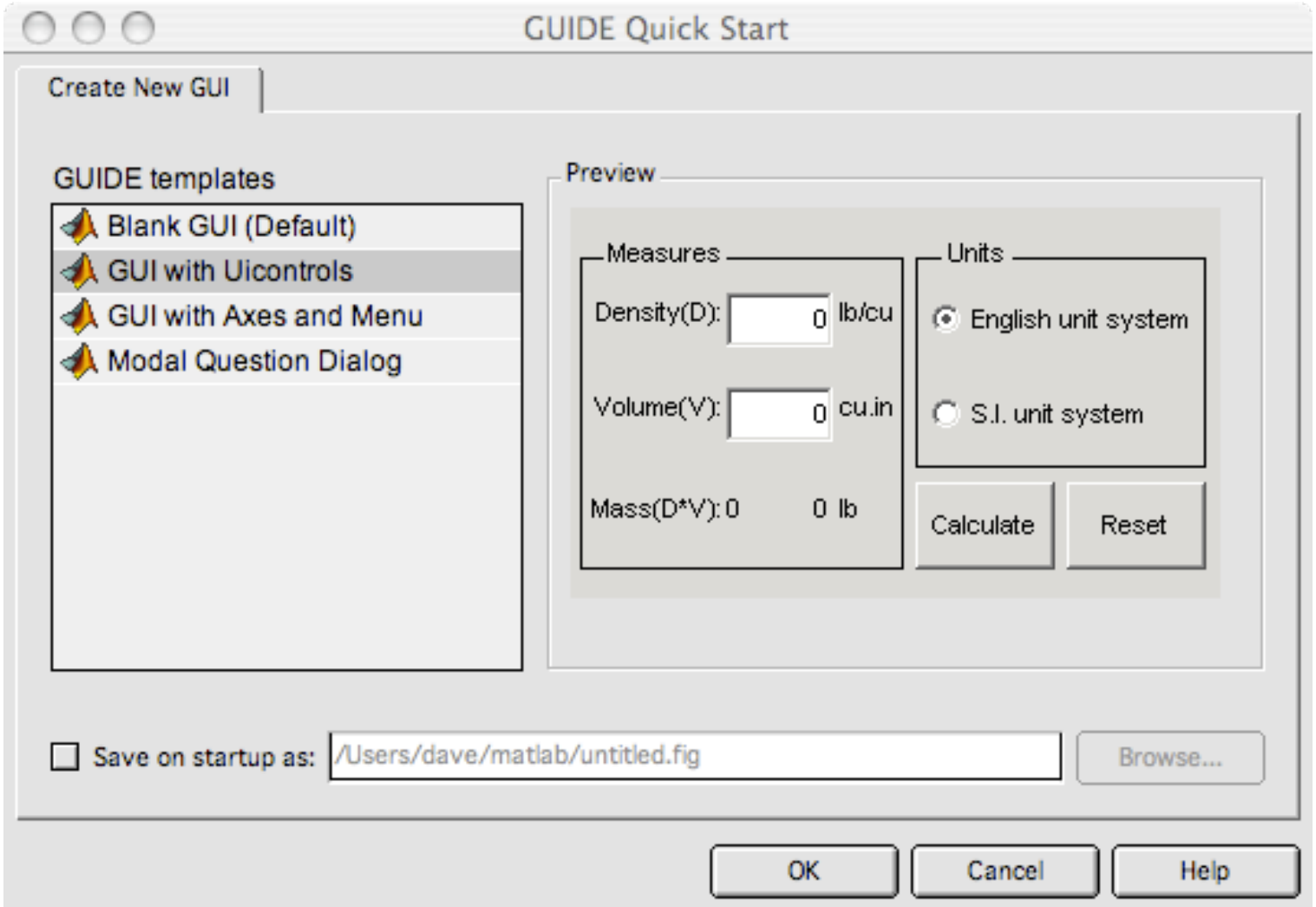
- Designing the overall layout and placement of UI elements is easy
- Editing UI element properties is easy
- Guide provides 4 templates with which to assemble your GUI:
  - A blank GUI (default)
  - GUI with Uicontrols
  - GUI with Axes and Menu
  - Modal Question Dialog
- Can also open existing GUIDE GUIs you have made

To invoke GUIDE: Type `guide` at command line.

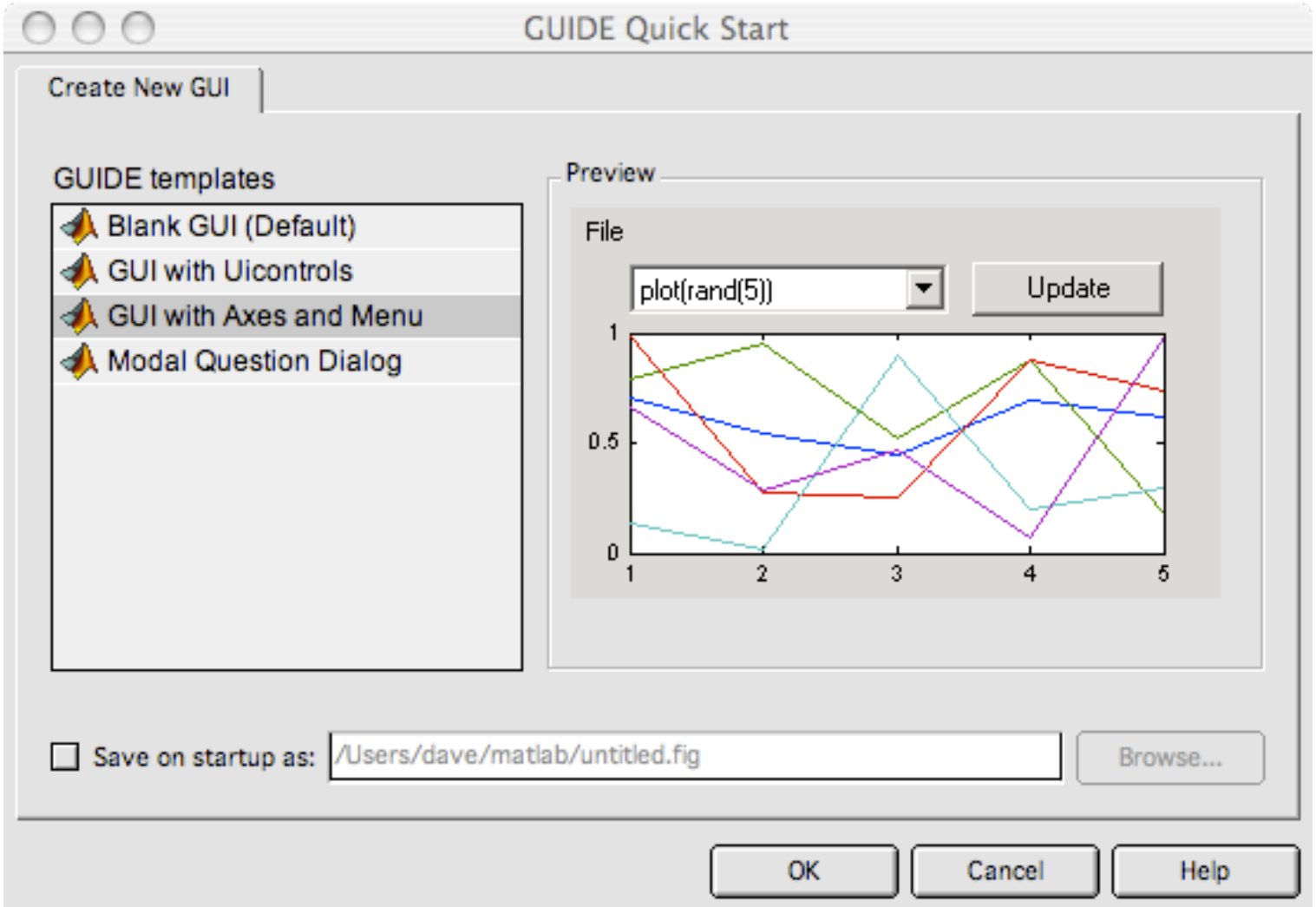
GUIDE: A blank GUI (default)



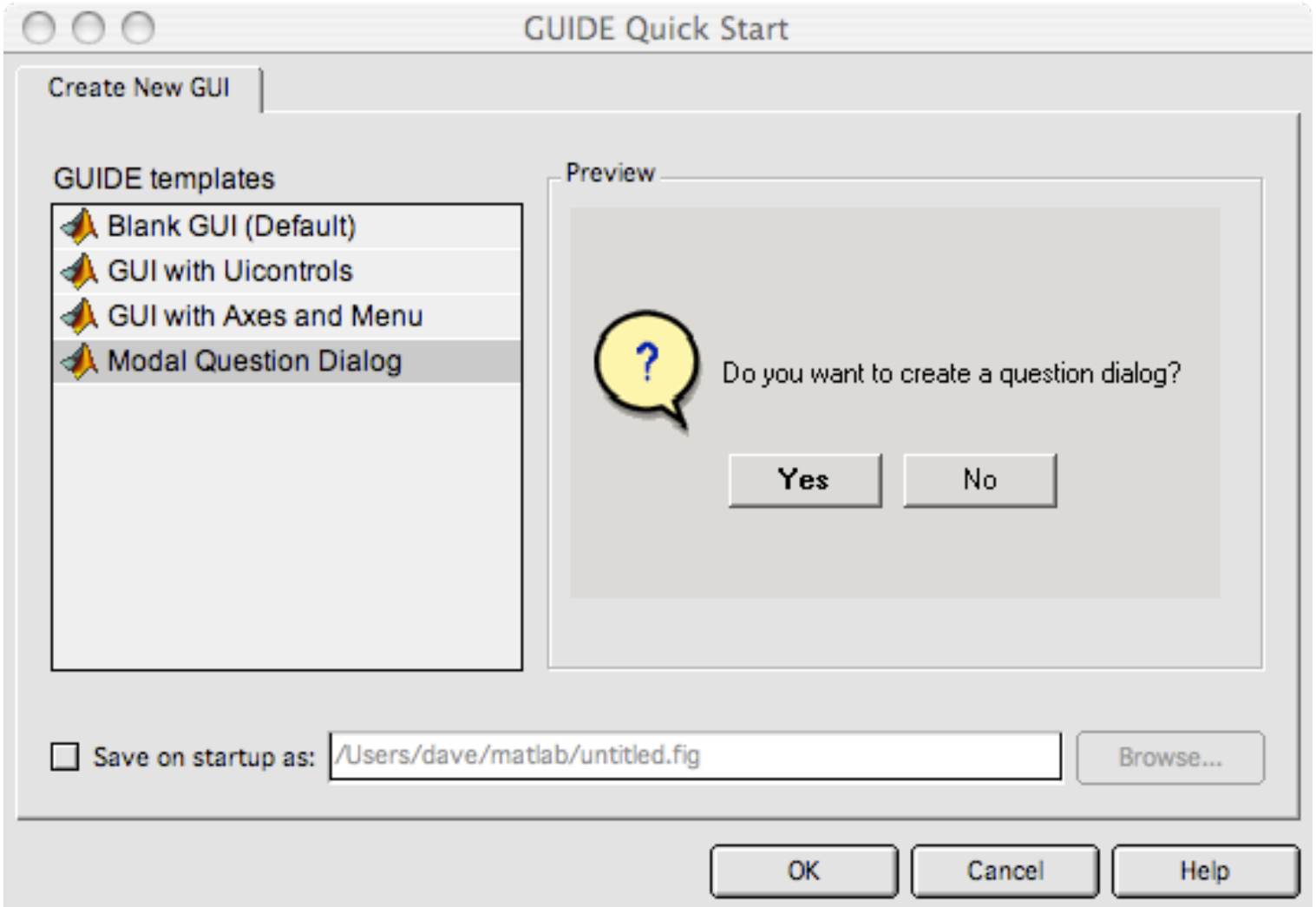
GUIDE: GUI with Uicontrols



GUIDE: GUI with Axes and Menu



GUIDE: Modal Question Dialog



GUIDE Layout Editor

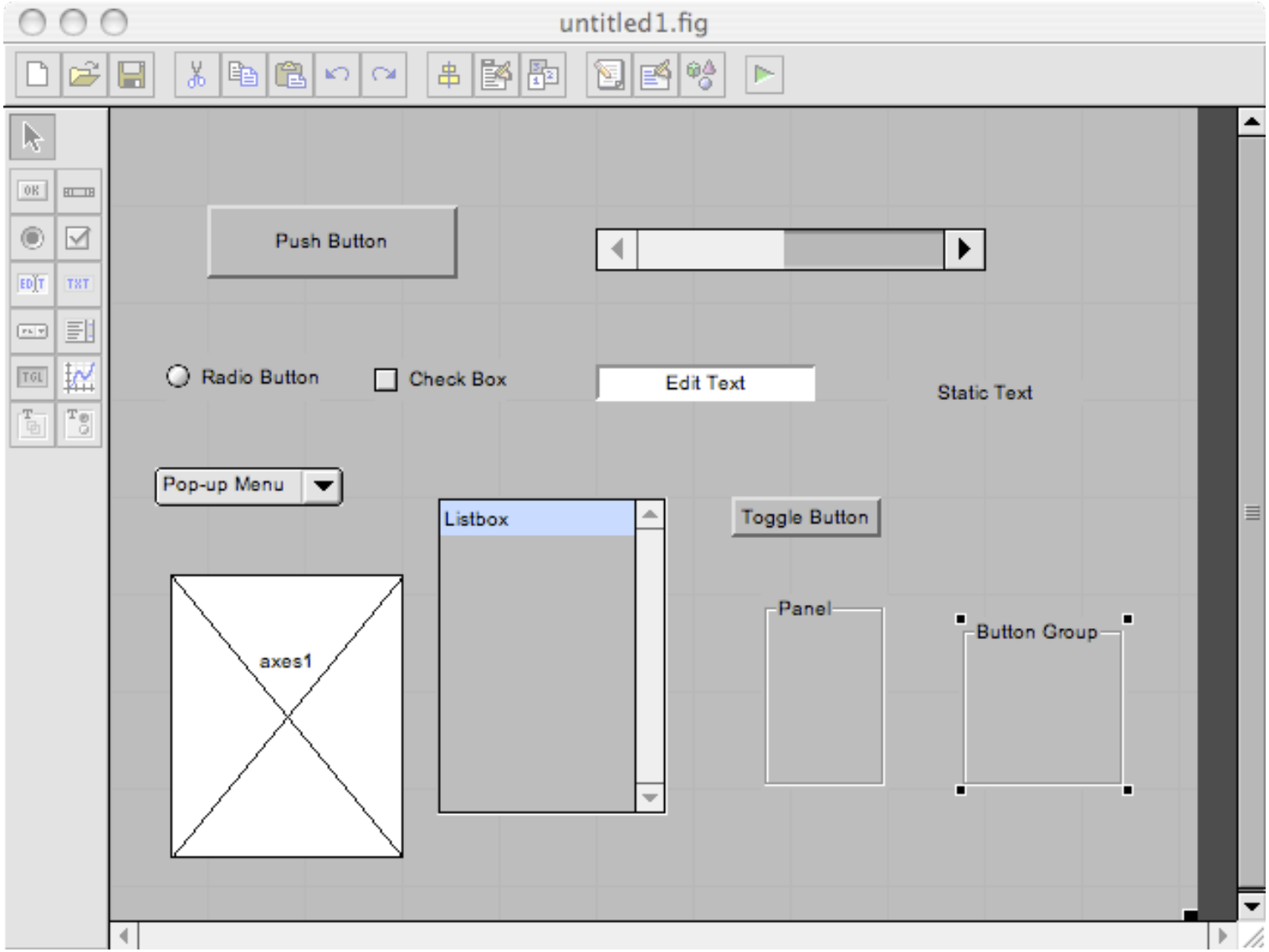
Whichever GUIDE template you select:

- Click on **OK** button in chosen template

You get the Layout Editor:

- Choose Uicontrol elements on the left panel
- Use **select arrow** to move/resize *etc.*
- Double click on any Uicontrol element to see **Property Inspector** to edit the element — **Example soon**

Layout Editor with sample Uicontrol Elements

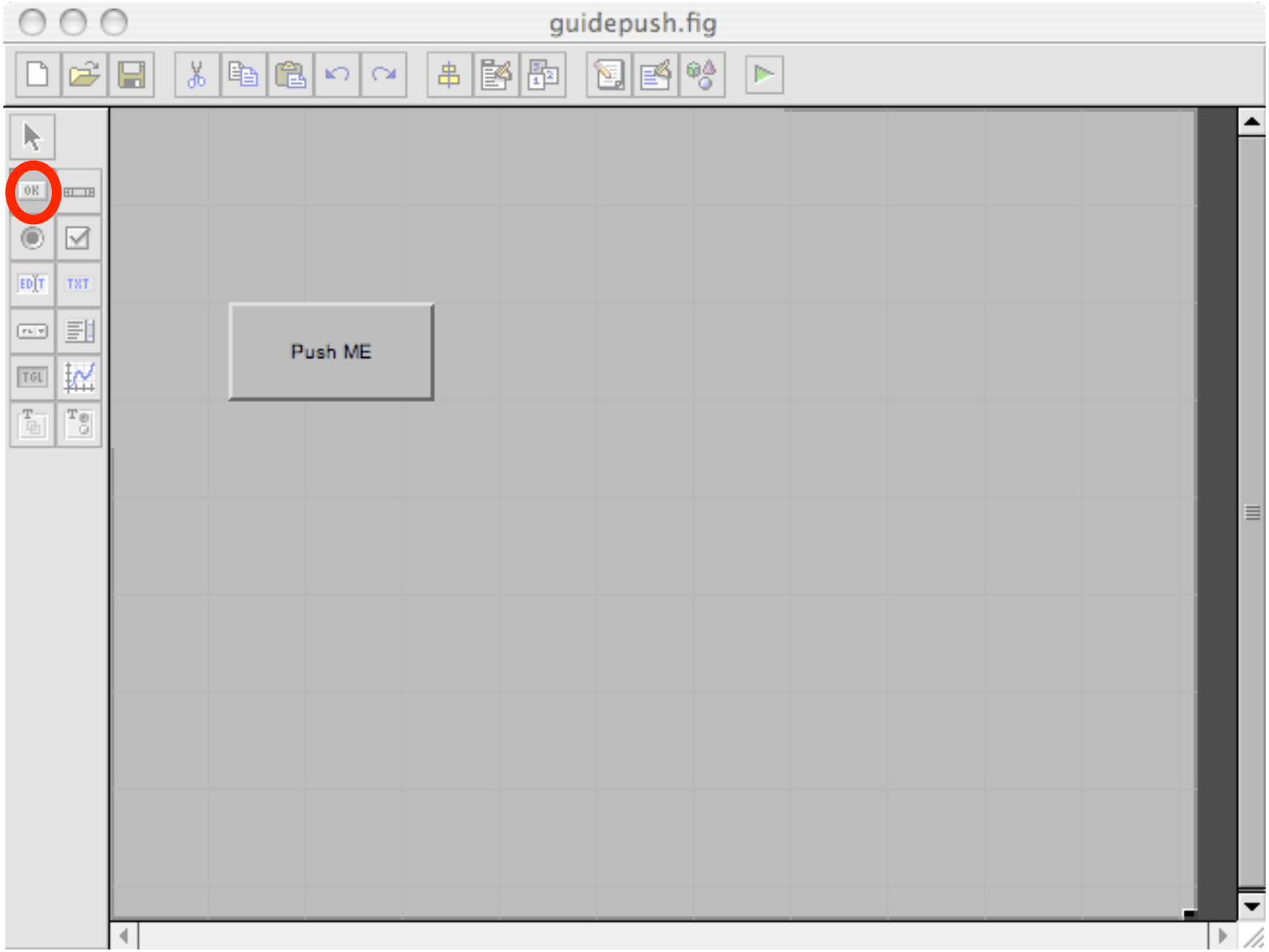


Creating a Simple GUI

Let's illustrate how we use GUIDE to create a simple push button GUI element:

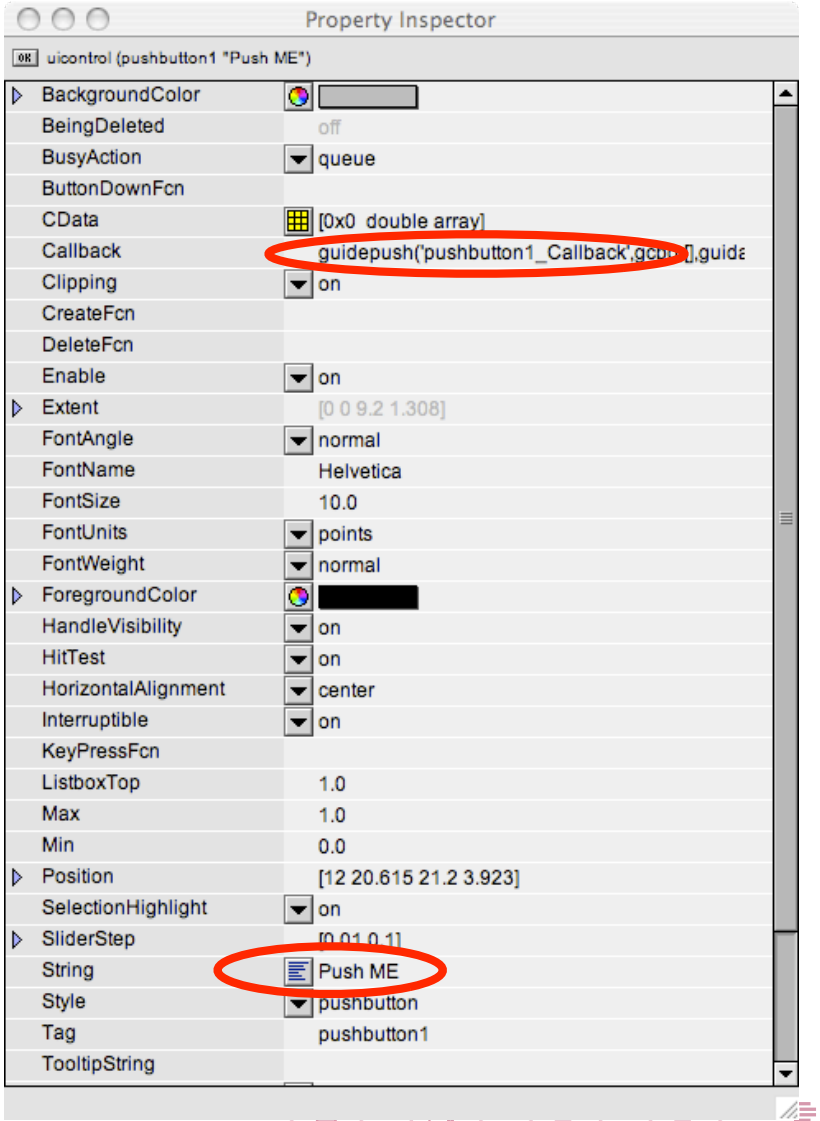
- Start GUIDE: Type `guide` at command line.
- Select a blank GUI template
- Click on OK Button
- Select a Push Button
- Draw a Push Button
- Double click on the button to invoke **Property Inspector**
- Change the buttons text from `Push Button` to `Push ME`.
- Save session as `guidepush`, for example. **Two files created**
  - `guidepush.m` — run this from the command lin
  - `guidepush.fig` — (binary format) GUI data, read by `guidepush.m`.

Example Push Button in Layout Editor



Push Button Property Inspector

- Note list of properties — useful for command programming reference
- **String** changed to `Push ME`
- Note callback function:
  - Can be changed
  - We edit this callback
- Other useful stuff to edit.



Adding Functionality to a GUIDE Callback

If you look at the guidepush.m file:

- Quite a lot MATLAB code
- **ONLY edit callback** — unless you know what you are doing
- Callback is pushbutton1_Callback()
- Let's add some simple functionality to this

```
function varargout = guidepush(varargin)
% GUIDEPUSH M-file for guidepush.fig
%      GUIDEPUSH, by itself, creates a new GUIDEPUSH or raises the existing
%      singleton*.
%
%      H = GUIDEPUSH returns the handle to a new GUIDEPUSH or the handle to
%      the existing singleton*.
%
%      GUIDEPUSH('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUIDEPUSH.M with the given input arguments.
%
%      GUIDEPUSH('Property','Value',...) creates a new GUIDEPUSH or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before guidepush_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to guidepush_OpeningFcn via varargin.

........

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guidepush_OpeningFcn, ...
                   'gui_OutputFcn',  @guidepush_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before guidepush is made visible.
function guidepush_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 guidepush (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

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

% --- Outputs from this function are returned to the command line.
function varargout = guidepush_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.


function pushbutton1_Callback(hObject, eventdata, handles)



% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Editing the Push Button Callback

Initially the callback has no functioning code:

- Let's add a simple print statement in traditional Hitchhikers Guide to the Galaxy mode:

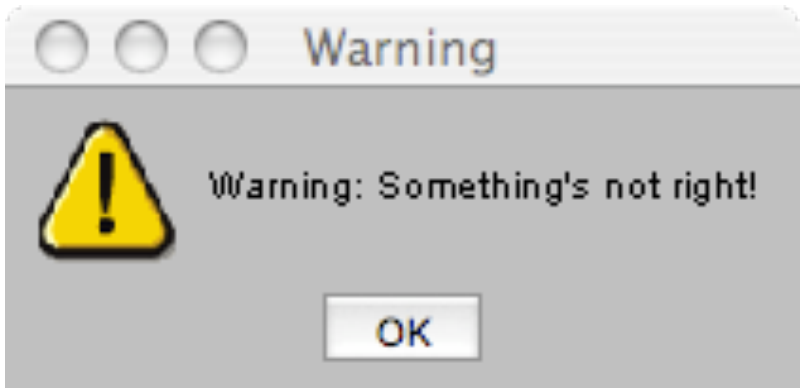
```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
% handles    structure with handles and user data (see GUIDATA)

disp('Dont Push Me!');
```

The Warning Dialog box: `warndlg`

To create a warning dialog you do something like this:

```
warnfig = warndlg('Warning: Something''s not right!', 'Warning');
```

This creates:



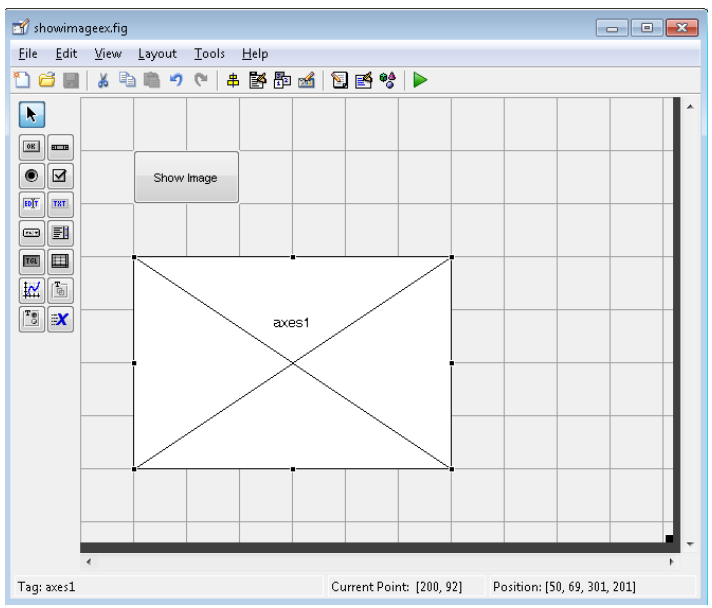
Note:

- The first string specifies the main error dialog text.
- The second string specifies the dialog window title text
- **Use** `''` to get a `'` character in a string

Interaction between Multiple Objects

MATLAB use *handles* to represent objects, including UI controls as well as graphics objects such as figures, axes, plots etc.

- Start GUIDE and add a **push button** and change the *String* property to "Show Image".
- Add an **axes** object and set the *Visible* to "off". Note the *Tag* property of the axes.
- Create the push button *Callback* as before.



Interaction between Multiple Objects (cont.)

Add the following code to the created *Callback* handler:

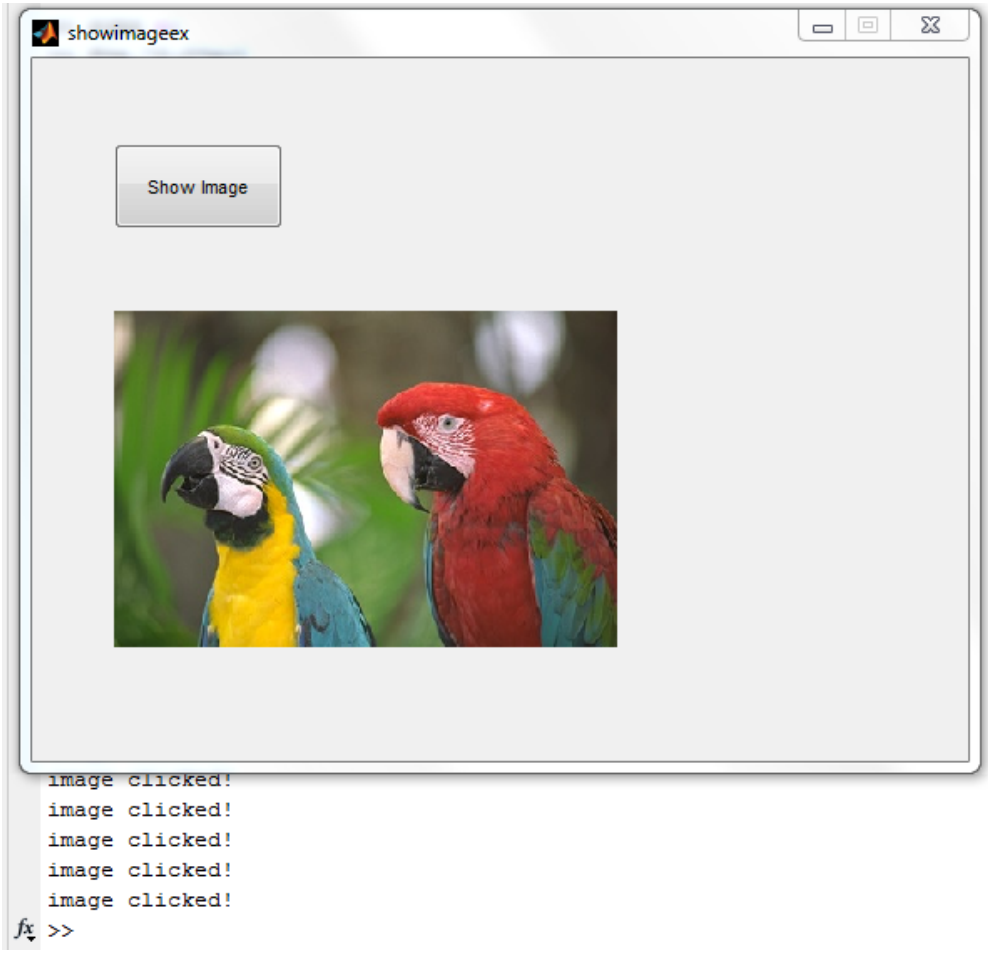
```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
im=imread('parrots.jpg');
himage = imshow(im, 'Parent', handles.axes1);
set(himage, 'ButtonDownFcn', @image_clicked);

function image_clicked(hObject, eventdata, handles)
disp('image clicked!')
```

- *handles* can be used to access the handle of other objects, e.g. *axes1*.
- **imshow** (and many other functions) take PropertyName, PropertyValue pairs, and the *Parent* property specifies the holder of the image. The return value of **imshow** is the handle to the image.

Interaction between Multiple Objects (cont.)

- **set** can be used to set the properties of an object, including event handlers (e.g. *ButtonDownFcn*).
- See the property inspector or **doc uicontrol** and then follow the **Uicontrol Properties** link.
- The event handler is a function of the similar format as the *Callback* function. Use @ followed by the function name to **set** the event handler. (A command string can also be used but only if the function is globally accessible.)

Interaction between Multiple Objects (cont.)



Clearly a lot more to GUIDE — check MATLAB built in docs and help and textbooks

MATLAB and Complex Numbers

MATLAB knows about complex numbers

```
>> sqrt(-1)
ans = 0 + 1.0000i

% Symbolic Eqns Soln
>> syms x;
>> f = x^2 + 1;
>> solve(f)
ans =

   i
  -i

% Polynomial Roots
>> p = [1 0 1];
>> roots(p)
ans =

        0 + 1.0000i
        0 - 1.0000i
```

Declaring Complex Numbers in MATLAB

Simply use *i* in an expression or the `complex()` function

```
% Must use * operator with i even though this is not displayed
>> c1 = 3 + 4*i
c1 =

   3.0000 + 4.0000i

% MATLAB also allows the use of j
>> c2 = 2 + 4*j
c2 =

   2.0000 + 4.0000i

% What I already have a variable i (or j)  e.g. for i=1:n?
>> c3 = complex(1,2)
c3 =

   1.0000 + 2.0000i
```

MATLAB: real, imaginary, magnitude and phase

MATLAB provides functions to obtain these

```
>> c = 4+3*i
c =
   4.0000 + 3.0000i

%  Real part,  Imaginary part,  and  Absolute value
>> [real(c),  imag(c), abs(c)]
ans =
     4     3     5

%   A complex number of magnitude 11 and phase angle 0.7 radians
>> z = 11*(cos(0.7)+sin(0.7)*i)
z =
   8.4133 + 7.0864i

%   Recover the magnitude and phase of "z"
>> [abs(z), angle(z)]
ans =
   11.0000    0.7000
```

MATLAB understands Trig. form of a complex number

From the last slide example:

You can declare in trig. form but MATLAB coverts to normal representation

```
%   Trig. Form: A complex number of
%   magnitude 11 and phase angle 0.7 radians
>> z = 11*(cos(0.7)+sin(0.7)*i)
z =
   8.4133 + 7.0864i

% So Need to use abs() and angle() to
%  Recover the magnitude and phase of "z"
>> [abs(z), angle(z)]
ans =
   11.0000    0.7000
```

MATLAB Complex Arithmetic

Behaves as one would expect

```
>> c1 = 3 + 4*i;
>> c2 = 2 + 4*j;
>> c1 + c2
ans =  5.0000 + 8.0000i

>> c1 - c2
ans = 1

>> i^2
ans = -1

>> c1*c2
ans = -10.0000 +20.0000i

>> c1/c2
ans = 1.1000 - 0.2000i
```

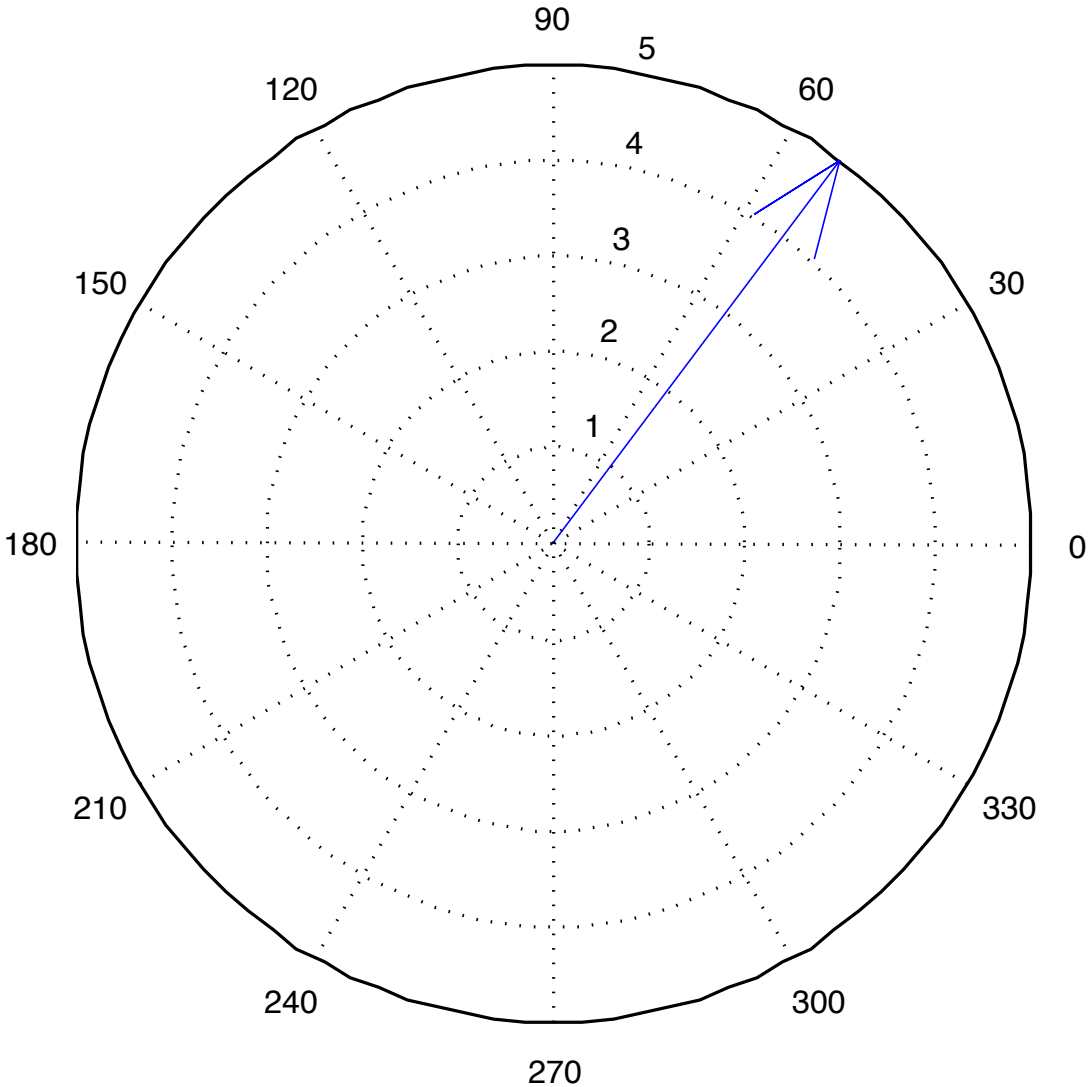
Plotting Polar Coordinates in MATLAB: compass() Plot

The compass() knows how to plot a complex number directly:

compass() Example

```
>> c1 = 3 + 4*i;
>> compass(c1);
```

`compass(c1);` Plot Output



Note: `c1` automatically converted to polar form

MATLAB and Phasors

MATLAB Complex No. Phasor Declaration

```
>> exp( i*(pi/4) )

ans =  0.7071 + 0.7071i

>> [abs(z), angle(z)]

ans =  1.0000    0.7854
```