Graphics Objects in Matlab

When we plot data in Matlab, we create a line object (and an axes object, and a figure object). The handle of the line object is the output argument for plot:

```matlab
a = linspace(0, 2*pi, 500);
b = sin(a);
H = plot(a, b);
```

The value of the variable `H` is the "handle of the line object" created by plot. The value is a unique identifier (an "address" or "pointer") of the specific line that was drawn:

```matlab
set(H, 'LineWidth', 4);
set(H, 'Color', [1 0 0]);
set(H, 'Visible', 'off');
set(H, 'Visible', 'on')
```

Creating graphics objects in different workspaces

Regardless of what is the "current" workspace when the object is created, the object is always accessible everywhere through its handle:

```matlab
a = 1:.01:10;
b = cos(a);
% function mkplot(x,y)
% mkplot(a,b);
```

After function exits, the line still exists, unlike the function workspace. Using get and the 'Children' property, we can get its handle:

```matlab
F = gcf % GetCurrentFigure
A = get(F, 'Children')
H = get(A, 'Children')
get(H, 'Type')
```

Globally accessible, hidden data

"Within each" graphics object, you can store hidden, globally accessible data:

```matlab
% Use the command setappdata.
% This is called application data.
Example
A = rand(40, 40);
setappdata(H, 'BigArray', A);
clear A
whos % H is still there
```

Delete the object using `delete`. The variable `H` whose value is the handle of the object remains, but is now just a scalar double:

```matlab
delete(H) % line object disappears
whos % H is still there
```

Retrieve the data using `getappdata`:

```matlab
tmp = getappdata(H, 'BigArray');
tmp(3, 2)
```

Data is "globally accessible" in that only the handle of the object is needed to access its application data.

Is there a `@handle` class?

For some reason, no. The variable returned by `plot` is simply an object of class `double`:

```matlab
H = plot(a, b);
class(H), size(H), H
```

So, the variable `H` is not the line object. The value of `H` is the address of the line object. We say the "value of `H` points to the line object" or just "`H` points to the object."

Clearing the variable does nothing to the line itself.

```
whos
clear H
whos
```

The variable is gone, but the line object remains...

Replot, for next slide

```matlab
H = plot(a, b);
```

Access the properties with the handle

Use `get` to discover the public properties of the object:

```matlab
PubProp = get(H);
get(H, 'Type')
get(get(H, 'Parent'), 'Type')
```

Fieldnames are the properties, and values are current values. Change the properties with `set`:

```matlab
set(H, 'LineWidth', 3);
Check workspace.
whos % H there, graphics objs not listed
```

Delete the object using `delete`. The variable `H` whose value is the handle of the object remains, but is now just a scalar double.

```matlab
delete(H) % line object disappears
```

Delete the object using `delete`. The variable `H` whose value is the handle of the object remains, but is now just a scalar double.

```matlab
delete(H) % line object disappears
whos % H is still there
```

```
get(H) % no graphics object
% associated with this value
```
**Summary: All graphical objects in Matlab**

- Are referred to by a **handle** (also called **pointer** or **address**)
- Have a **get/set** interface by which their properties are accessed and modified
- Are created by **constructor** methods, but are not variables
  - Not being variables, they are not in any workspace
  - Being created with constructors, they are created from a specific workspace, but exist independent of workspace.
- Can store hidden, globally accessible data, called **application data** (in the form of Matlab variables) within them that exists as long as the object itself exists. By “within them” we mean
  - Variables are not in any workspace
  - Variables exist as long as the object exists
  - Variables are accessed/deleted/etc by knowing the object’s handle
- Have **events** associated with them
  - Example: a pushbutton object has a ButtonPress event; a Figure object has a MouseMotion event, etc.
  - Programs, called “callbacks” can be associated with events so that when the event occurs, the callback program is executed

**Graphical objects .vs. Variables**

Contrast the attributes of the graphical objects with the attributes of Matlab variables (like **double**, **cell**, **char**, **struct** and objects of user-defined classes)

**Graphics Objects:**
- Are referred to by an address, called the “handle”
- Have a get/set interface to access/modify
- Are not associated with a workspace
- Can hide data
- Can generate events

**Matlab variables:**
- Are referred to by name
- Are reference directly to access/modify (primitives)
- Have a get/set interface to access/modify (objects)
- Live in a workspace
- Cease to exist when the workspace they live in is deleted.

**Root Object**

Not really a graphics object... but behaves as one

**Constructor**
- There is none, object is created when Matlab is started

**Properties (try get(0) and set(0) to discover them)**
- Diary, DiaryFile
- Format, FormatSpacing
- PointerLocation, PointerWindow
- RecursionLimit

---

**Button press events for line objects**

If the mouse is pressed while the pointer is over the line
- A ButtonDownFcn event occurs, so...
  - If a ButtonDownFcn callback has been set, it is executed
The **callback** may be set in two manners
- Using a char
- Using a cell

In the case of a char, the string is evaluated in the base workspace

```matlab
cbstr = ['disp(''Mouse pressed'');'];
set(H,'ButtonDownFcn',cbstr);
```

Now, if the mouse is pressed over the line, the command `eval(cbstr)` will be executed in the base workspace, i.e., as though you typed

```matlab
>> eval(cbstr)
```

Do it a few times, and it works.

Easy to understand, but often not flexible enough... More generally, cell arrays will be used to specify callbacks.

---

**Graphics Objects in Matlab**

Some of the different types of graphics objects in Matlab are

```
Matlab session, Handle equals 0
the Root object
figure
child of root
...
axes uicontrol uimenu uicontextmenu
child of figure
...
line patch surface text
child of axes
```
The `figure` Objects

- Constructor: `figure` automatically constructed when creating an `axes`, `uicontrol`
- Properties (lots...):
  - Position
  - Color
  - CurrentPoint
  - HandleVisibility
- Events/Callbacks:
  - KeyPressFcn
  - CloseRequestFcn
  - ResizeFcn
  - WindowButtonDownFcn
  - WindowButtonMotionFcn
  - WindowButtonUpFcn

The `axes` Objects

- Constructor: `axes`
- Properties:
  - CameraPosition
  - CurrentPoint
  - NextPlot
  - XTick, XTickLabel, XScale, XLim, XGrid, XDir
  - YTick, YTickLabel, ...
  - ZTick, ZTickLabel, ...
  - ...
- Events/Callbacks:
  - ButtonDownFcn
  - DeleteFcn

The `uicontrol` Objects

- Constructor: `uicontrol`
- Properties:
  - Style: checkbox, pushbutton, edit, text, frame, popupmenu, listbox, radiobutton, slider, togglebutton
  - ToolTipString
  - Value
  - Enable
- Events/Callbacks:
  - Callback
  - DeleteFcn
  - ButtonDownFcn

Position and Units properties

Several objects (figure, axes, uicontrol) have properties named `Position` and `Units`. How are these interrelated?

- If `F` is the handle of a `figure`, then `get(F,'Position')` is the position (LLX,LLY,Width,Height) relative to the LL corner of screen, expressed in the units of `get(F,'Units')`
- Note that `get(0,'Units')` plays no role.

- If `A` is the handle of a `axes`, then `get(A,'Position')` is the position (LLX,LLY,Width,Height) relative to the LL corner of parent `figure`, expressed in the units of `get(A,'Units')`
- Again, `get(A,'Parent'),'Units')` plays no role.

CurrentPoint properties

- If `F` is a `figure` handle, then `get(F,'CurrentPoint')` is the pointer location (X,Y) relative to the LL corner of screen, expressed in the units of `get(F,'Units')`
- If `A` is an `axes` handle, then `get(A,'CurrentPoint')` is a 2x3 array of the “position” of the pointer, in the coordinates of the axes.
- Take line perpendicular to screen, passing through pointer.

```
\begin{bmatrix}
X_{front} & Y_{front} & Z_{front} \\
X_{back} & Y_{back} & Z_{back}
\end{bmatrix}
```

- Note that `get(0,'Units')` plays no role.

Both are updated on `buttonclick` events, such as `ButtonDownFcn`, `WindowButtonMotionFcn`
Writing reusable graphical based applications

In order to write a reusable, graphical based application using Matlab HandleGraphics, the "application" should mimic the attributes of the basic Matlab HandleGraphics objects
- referred to by a handle
- get/set interface by which its properties ("state") are accessed
- store hidden, globally accessible data
- have events associated with changes in its state

To accomplish this, it is best to integrate the graphics objects with the user defined classes. We will do this, over the next two weeks.

Today, we give a partial overview of the strategy, minus the object oriented (user defined classes) wrapper.

We need one more Matlab concept before starting
- Cell array callbacks (as opposed to char callbacks)

Cell array callbacks

Create (for example) a pushbutton

\[ PB = uicontrol('style','pushb'); \]

and a function, ARG5.m

```matlab
function ARG5(A1,A2,A3,A4,A5)
... % some code here
... % some code here
ARG5.m
```

and a 1x4 cell array

\[ cb = {{\text{ARG5}}{-14.1}{'e177'}[100;10;1]}; \]

Set the callback of the pushbutton to be the cell array.

\[ set(PB,'Callback',cb); \]

What happens when the button is pressed?

Cell array callbacks, common mistake

Setup is

\[ PB = uicontrol('style','pushb'); \]
\[ V = 17 \]
\[ cb = {{\text{ARG5}}{-14.1}{'e77'}V}; \]
\[ set(PB,'Callback',cb); \]
\[ V = 18; \]
\[ cb = {{\text{ARG5}}{-14.1}{'e77'}V}; \]

Now press button. What will the value of the 5th argument to the function ARG5 be?

Remember, Matlab will effectively execute (in a private workspace)

\[ tmp = gcbo('Callback'); \]
\[ feval(tmp{1},EventSrc,EventData,tmp{2:end}) \]

Clarify: not actually the variable \( cb \), but the contents of pushbutton’s Callback property, \( \text{get(PB,'Callback')} \)

PushButton/Counter

Let's make a reusable object consisting of a
- counter (initialized at 0)
- pushbutton (to increment counter)
- text display of counter value

Initial appearance

\[ ++ 0 \]

After one buttonpress

\[ ++ 1 \]

After 7 buttonpresses

\[ ++ 7 \]

User should also be able to program additional actions to occur after a button press.

Overview of Tool Code

```matlab
function ToolH = fname(arg)
% Create objects which constitute the tool
% These are the "subobjects"
% Choose one subobject as the "tool handle" (ToolH)
% Create ToolState struct which holds tool "state"
% Hide ToolState in appdata of ToolH
% Set subobject properties, event callbacks to subfcns
% Input arg: ToolH (then subfcns can access ToolState)

function subf1(Esrc,Edata,ToolH)
% Ignore Esrc, Edata
% Retrieve ToolState (from appdata of ToolH)
% Do calcs, set properties of subobjects
% Update and replace ToolState (appdata of ToolH)

function subf2(Esrc,Edata,ToolH)
function subf3(Esrc,Edata,ToolH)
```
PushButton/Counter State

Graphics object is the counter/display

What information do we need to keep track of everything?
– value of counter
– handle of text display
– function_handle that user wants to additionally execute when the value is incremented

Hence, tool state will be a structure with 3 fields
– CntVal, integer counter value
– DispHan, the handle of the text display uicontrol
– IncCB, function_handle (plus additional arguments) of user defined increment callback

```
function ToolHan = e177gui1(X,Y,IncCB)
  FrameH = uicontrol('style','frame',...
                'position', [X Y 110 30]);
  PlusH = uicontrol('style','pushbutton',...
                'position', [X+5 Y+5 40 20], 'str','++');
  DispH = uicontrol('style','text',...
                'position', [X+50 Y+5 40 20]);
  ToolHan = DispH;
  ToolState.DispHan = DispH;
  ToolState.CntVal = 0;
  ToolState.IncCB = IncCB;
  setappdata(ToolHan,'ToolState',ToolState);
  set(PlusH,'callback',{@LOCALinc ToolHan});
  set(DispH,'str',int2str(ToolState.CntVal));

function LOCALinc(ESrc, EData, TH)
  TS = getappdata(TH,'ToolState');
  TS.CntVal = TS.CntVal + 1;
  set(TS.DispHan,'str',int2str(TS.CntVal));
  setappdata(TH,'ToolState',TS);
  if ~isempty(TS.IncCB)
    feval(TS.IncCB{1},ESrc,EData,...
          TH,TS.IncCB{2:end})
  end
```

PushButton Demo

guidemodemoA.m

```
x = linspace(0,2*pi,200);
Linel = plot(x,sin(0*x));
set(Linel,'LineWidth',3);
H1 = e177gui1(10,05,{@replot Linel});
```

replot.m

```
function replot(Esrc,Edata,CntTool,LH)
  YD = get(LH,'Ydata');
  XD = get(LH,'Xdata');
  TS = getappdata(CntTool,'ToolState');
  set(LH,'Ydata',sin(TS.CntVal*XD));
```

PushButton Demo 2

guidemodemoB.m

```
x = linspace(0,2*pi,200);
Linel = plot(x,sin(0*x));
set(Linel,'LineWidth',3);
H1 = e177gui1(10,05,{@replot Linel});
H2 = e177gui1(160,05,{@wacky Linel});
```

wacky.m

```
function wacky(Esrc,Edata,CntTool,LH)
  set(LH,'Color',rand(1,3));
```