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 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 referenced 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.

Our Strategy thusfar

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 subfcs
% Input arg: ToolH (then subfcs 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)

In reality, ToolH is simply the handle to one of the graphics objects
So, back in the base workspace, get(ToolH) just shows it to be a single HandleGraphics object.

Private Data and Application Data

Facts:
An object needs to hide its state (private database)
Programmers who employ this object may want to hide their own data within the object (“application” or “programmer” database)

Analogy: uicontrol (‘style’, ’pushbutton’)
- The button’s states are stored somewhere, which we cannot see
  - With get and set, users (us) can alter the properties, which ultimately involve hidden changes in the state.
  - We, as users, are allowed to hide data “inside” the uicontrol using the commands setappdata and getappdata

Solution: Every Object will have a private database.
- The object itself hides its state in the private database
- The object maintains an application database (appdata) for users of the object
- Commands to access these are
  setappdata(H,’Name’,Value) % used in object’s methods
  setappdata(H,’Name’,Value) % for users of object
  getappdata(H,’Name’,Value)
Listeners
Associated with an object, a listener associates a callback
– to a property and actions that affect the property

Example:

Object Y should read the value, and set Y’s FrequencyMultiplier to that same value.

Essentially, a listener for object X of class A is a structure, usually stored in the private database of X, with fields

- .Property, a valid property for objects of class A
- .Action, the action that is “listened” for, such as “Set”
- .Callback, a standard cell-array callback declaration

In this case, if the listener is defined in object X, then the callback is executed whenever the specified property of X is set.
This allows another object Y, of any class, to “register” a listener with X, so that Y is “notified” when the property of X is changed.

The @classname folder
If a class inherits hgtoolkit, then it must have the following 3 methods

- fnames = fieldnames(P) % returns cell
- Value = pvget(P,PropertyName) % private GET
- pvset(P,PropertyName,Value) % private SET

In the Inheritance lectures, we envisioned a more complex set of private access routines. Those ideas could/should be followed here, but cloud the issue. Today, limit our discussion as described.

Their functionality must follow standards, given next.

Our New Strategy

function P = classname(varargin)
% Create objects ('the subobjects') which constitute the tool
H1 = uicontrol('style','pushbutton');
...

% Make structure with one field: cell array of handles of all subobjects
P.ObjectList = {H1;H2;...};

% Make structure a class, and inherit all methods from hgtoolkit
P = class(P,'classname',hgtoolkit);
% Initialize the object (creates ProgrammerDB and ListenerStruct)
initialize(P);
% Set subobject properties
set(H1,PropertyName,Value);
% Set object state using SETPVDBDATA
setpvdbdata(P,PropertyName,Value);
% Add listeners to subobjects
addlisteners(H1,Counter,'PostSet',{@LOCALsubf1 P varargin},0);
...
% Undo all of the default settings
set(P,varargin{:});

function LOCALsubf1(Esrc,Edata,P,args)
% Ignore Esrc, Edata
Use set, get, on subobjects
Use get, set on P
Use setpvdbdata, getpvdbdata on P

Private GET function, written specifically for object
- accesses PrivateDB of object, and/or
- uses get to access subobject properties.

function [value] = pvget(P,property)
switch property
  case Case1
    value = getpvdbdata(P,property);
  case Case2
    value = get(P.ObjectList{3},'visible');
  ...
  case 'PropNames'
    value.GPropNames = {...}; % gettable
    value.SPropNames = {...}; % settable
  case 'ObjectList'
    value = P.ObjectList
  otherwise
    error([property ' is Invalid']);
end

Private SET function, written specifically for object
- alters PrivateDB of object, and/or
- uses set to alter properties of subobjects

function pvset(P,property,value)
switch property
  case Case1
    setpvdbdata(P,property,value);
  case Case2
    set(P.ObjectList{3},'visible','on');
  otherwise
    error([property ' is Invalid']);
end

Simple constructor, no fields for now. Any class that inherits hgtoolkit will have access to all of its methods.

function h = hgtoolkit
h = class(struct({}),'hgtoolkit');
@hgtoolkit/initialize.m
Purpose: Creates two structs in the appdata of the first entry of ObjectList.

```matlab
function initialize(P)
    H = pvget(P,'ObjectList');
    PDB = struct([],{});  % empty structure
    LS = struct('Property',cell(0,1),...
                  'When',cell(0,1),...
                  'Callback',cell(0,1),...
                  'GroupName',cell(0,1),...
                  'UniqueID',cell(0,1),...
                  'Enable',cell(0,1));  % 0-by-1 struct
    % APPDATA of first object in ObjectList
    % stores ProgrammerDB, and ListenerStruct.
    setappdata(H{1},'ProgrammerDB',PDB);
    setappdata(H{1},'ListenerStruct',LS);
```

@hgtoolkit/setpvdbdata.m
Purpose: set PrivateData of an Object

```matlab
function setpvdbdata(P,property,value)
    H = pvget(P,'ObjectList');
    pvdbhan = H{1};
    setappdata(pvdbhan,property,value);
```

@hgtoolkit/getpvdbdata.m
Purpose: retrieve PrivateData of an Object

```matlab
function value = getpvdbdata(P,property)
    H = pvget(P,'ObjectList');
    pvdbhan = H{1};
    if nargin==2
        if isfield(value,property)
            value = getfield(value,property);
        else
            warning(['Could not find ' property]);
            value = [];
        end
    end
```

@hgtoolkit/getappdata.m
Purpose: Retrieve ApplicationData of an Object

```matlab
function value = getappdata(P,property)
    value = getpvdbdata(P,'ProgrammerDB');
    if nargin==2
        if isfield(value,property)
            value = getfield(value,property);
        else
            warning(['Could not find ' property]);
            value = [];
        end
    end
```

@hgtoolkit/setappdata.m
Purpose: Set ApplicationData of an Object

```matlab
function setappdata(P,property,value)
    tmp = getpvdbdata(P,'ProgrammerDB');
    if isempty(tmp)
        tmp = struct(property,value);
    else
        tmp = setfield(tmp,property,value);
    end
    setpvdbdata(P,'ProgrammerDB',tmp);
```

Design decision: appdata of an object is stored in a structure, called ProgrammerDB, which itself lies in the private database of the object (appdata of first subobject in ObjectList).

Interplay in set*data/get*data methods

- `setappdata(P,...)` calls `setpvdbdata(P,...)`
- `setpvdbdata(P,...)` calls `setappdata(P.ObjectList{1},...)`
- Looks circular, but its not.

At some point, this is a handle of a Matlab HandleGraphics object, and the builtin `setappdata` does the real work.
@hgtoolkit/set.m (pseudo code)

function set(P, varargin)
% Split VARARGIN in a PropList, and ValueList
for i = 1:length(varargin)
...
end
% Run any "PreSet" listeners
runlisteners(P, PropList, 'PreSet')
% Call PVSET to do the actual SET
pvset(P, 'PropList', ValueList)
% Run any "PostSet" listeners
runlisteners(P, PropList, 'PostSet')

@hgtoolkit/runlisteners.m
Purpose: Execute callbacks of enabled listeners associated with a given property, and action.

function runlisteners(P, prop, when, excludegroups)
LS = getpvdbdata(P, 'ListenerStruct');
if length(LS) > 0
% Find ENABLED Listeners assoc. with PROPERTY/WHEN
PropList = strcmpi(LS.Property, prop, length(prop));
WhenList = strcmpi(LS.When, when, length(when));
Enable = [LS.Enable];
for idx = find(PropList & WhenList & Enable)
% Check if listener LS(idx) is not excluded
if ~ismember(LS(idx).GroupNumber, excludegroups)
cb = LS(idx).Callback;
feval(cb{1}, P, LS(idx).UniqueID, cb{2:end});
end
end
end

@hgtoolkit/addlistener.m
Purpose: Execute callbacks of enabled listeners associated with a given property, and action.

function [ListenerID] = ...
addlistener(P, property, when, cb, groupnumber)
LS = getpvdbdata(P, 'ListenerStruct');
nlS = length(LS);
UID = [LS.UniqueID];
listenerID = min(setdiff(1:nlS+1, UID));
idx = nlS+1;
LS(idx).Property = property;
LS(idx).When = when; % 'PreSet' or 'PostSet'
LS(idx).Callback = cb;
LS(idx).GroupNumber = groupnumber;
LS(idx).UniqueID = listenerID;
LS(idx).Enable = 1;
setpvdbdata(P, 'ListenerStruct', LS); end

@hgtoolkit/eventtrigger.m
Recall how a tool might coordinate the three callbacks
WindowButtonDownFcn
WindowButtonMotionFcn
WindowButtonUpFcn
to achieve some affect, as in grabmousemotion. If another tool in the same figure needed to do similar coordination, the current approach breaks. The tools need to "know" about one another, ruining the decoupled viewpoint.

An alternative is to allow objects to add Listeners for
WindowButtonDown Events
WindowButtonMotion Events
WindowButtonUp Events, etc.,
and create a @superfigure class, which generates such Events when the mouse is pressed, moved, etc.,

@hgtoolkit: Other Listener functions
A few other commands, all self-explanatory, manage the properties of listeners within an object.

enablelistener(Obj, ListenerID)
disablelistener(Obj, ListenerID)
removelistener(Obj, ListenerID)
getlisteners(Obj)

Remark: The Listener concept presented here is quite elementary. I implemented it with a structure, and did not use classes/objects. It would be easy (and probably productive) to do so, yielding a nicer interface.

@hgtoolkit/eventtrigger.m
Purpose: gateway callback for non-"set" events to run listeners

function eventtrigger(Exrc, Edta, P, property, when)
runlisteners(P, property, when)

Problem: eventtrigger is a @hgtoolkit method, and its handle is generally unavailable to objects that need it to set a callback.

Solution: a @hgtoolkit method that simply returns the handle

function etHandle = ethan(P)
etHandle = @eventtrigger;
@superfigure/superfigure.m

Purpose: Give a figure the ability to generate Events that listeners can respond to.

```matlab
function P = superfigure(varargin)
    f = figure;
pv = 1:length(varargin);
P.ObjectList = {f};
P = class(P,'superfigure',hgtoolkit);
initialize(P);
setpvdbdata(P,'FigureHandle',f);
ethan = etHandle(P);
set(f,'ButtonDownFcn',{ethan P 'ButtonDown' 'Event'});
... % same with WindowButtonDownFcn, etc.
set(f,'KeyPressFcn',{ethan P 'KeyPress' 'Event'});
if length(pv>0)
    set(P,varargin{pv});
end
```

@superfigure/pvset.m

Purpose: Adhere to the @hgtoolkit rules, but give a superfigure object access to all of the properties that Matlab builds into figure objects.

```matlab
function [value] = pvset(m,property,value)
    switch property
        case {'ButtonDownFcn' 'WindowButtonDownFcn' ...
                'WindowButtonMotionFcn' 'WindowButtonDownFcn' ...
                'ResizeFcn' 'KeyPressFcn'};
            error(['Cannot set ' property ', use LISTENERS ']);
        otherwise
            fhan = getpvdbdata(m,'FigureHandle');
            set(fhan,property,value);
    end
```