Learning how to ...

GUI #2

!!! Attention: CHANGES IN GRAPHICS SINCE MATLAB R2014b !!!
Techniques of GUI design - sorting

- there exist several approaches (methodologies) to create GUI
  - design using GUIDE tool
    - not recommended
  - design using App Designer (from R2016a)
    - new graphic objects ('old' objects are not supported)
  - switch-board technique
    - not recommended
  - utilization of side and nested functions as callback functions
    - standard
  - fully OOP approach (including functional part of the code)
    - ideal
Callback function

- there are user-evoked events defined related to each object (button click, list selection, ...)
- these events are served by so called callback functions
  - in other words, when user pushes button, callback function of this event is activated (if defined).
- when GUI is not to be static, it has to contain at least one callback function
- callback function is stored as an object property – it is possible to change it, delete it, copy it etc.
Evaluation of callback function

- callback function is evaluated as a handle function

```
hb = uicontrol('Style', 'pushbutton', 'String', 'Plot line')

% Calling function using handle function
set(hb, 'Callback', @myFunc)

function myFunc(hObject, callbackdata)
% Callback function always adds two basic inputs

hObject       % reference to the object raising the callback
callbackdata  % object describing event
```
Evaluation of callback function

- callback function is evaluated as an anonymous function

```matlab
hb = uicontrol('Style', 'pushbutton', 'String', 'Plot line')

% TIP – anonymous function can be used in the case of calling a function that doesn't support basic inputs of callback function
set(hb, 'Callback', @(src, event)myFunc(inp))

function myFunc(inp)
    inp % the input are only variables defined by user
```
Evaluation of callback function

- callback function is evaluated as a handle function

```matlab
hb = uicontrol('Style', 'pushbutton', 'String', 'Plot line')

% Cell array, where first element is a handle function
set(hb, 'Callback', {@myFunc, inp1, ..., inpN})

function myFunc(hObject, callbackdata, inp1, ..., inpN)
% Basic inputs added to first positions again

hObject % reference to the object raising the callback
callbackdata % structure of various events (can be empty)

inp1, ..., inpN % other inputs
```
Evaluation of callback function

- **Ex.:** change background color of push button and change its label to 'Done' when clicked

```matlab
function GUI
close all
hButt = uicontrol('Units', 'normalized', 'Style', 'pushbutton', 'String',...
    'pushbutton', 'ForegroundColor', 'white', ...
    'BackgroundColor', [0.7 0.2 0], 'FontWeight', 'bold', ...
    'FontSize', 11, 'Position', [0.1 0.65 0.15 0.1]);
hButt.Callback = @pressButton;
end

function pressButton(scr, event)
% scr and event are default parameters returned by callback functions
% scr  - callback source (button handle object in this case)
% event - info on event raised (sometimes usefull)

disp(scr);   % check list - object handle
disp(event); % show info on raised event
set(scr, 'String', 'Done', 'BackgroundColor', rand(1, 3));
end
```
Evaluation of callback function

- callback function is evaluated as a string

```matlab
hb = uicontrol('Style', 'pushbutton', 'String', 'Plot line')

hb.Callback = 'plot(rand(20,3))';
```

- very limited possibilities
  - the string can contain variables as well
    - only the variables from base Workspace are evaluated correctly
  - GUI is usually created in functions
    - source handle object is not available
    - it is possible to call just scripts or main functions with predefined inputs
## Callback functions – list

<table>
<thead>
<tr>
<th>Callback</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>CellEditCallback</td>
<td>uitable</td>
</tr>
<tr>
<td>CellSelectionCallback</td>
<td>uitable</td>
</tr>
<tr>
<td>ButtonDownFcn</td>
<td>axes, figure, button group, panel, uiobjects</td>
</tr>
<tr>
<td>ClickedCallback</td>
<td>push tool, toggle tool</td>
</tr>
<tr>
<td>CreateFcn, DeleteFcn</td>
<td>axes, button group, context menu, figure, menu, panel, uiobjects, …</td>
</tr>
<tr>
<td>OffCallback, OnCallback</td>
<td>toggle tool</td>
</tr>
<tr>
<td><strong>ResizeFcn (&lt;R2014b)</strong></td>
<td>figure, panel, button group</td>
</tr>
<tr>
<td>SelectionChangeFcn</td>
<td>button group</td>
</tr>
<tr>
<td>KeyPressFcn</td>
<td>figure, uiobjects</td>
</tr>
<tr>
<td>KeyReleaseFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowButtonDownFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowButtonMotionFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowButtonUpFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowKeyPressFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowKeyReleaseFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowScrollWheelFcn</td>
<td>figure</td>
</tr>
<tr>
<td>CloseRequestFcn</td>
<td>figure</td>
</tr>
</tbody>
</table>
# Callback functions – list

<table>
<thead>
<tr>
<th>Callback</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>CellEditCallback</td>
<td>uitable</td>
</tr>
<tr>
<td>CellSelectionCallback</td>
<td>uitable</td>
</tr>
<tr>
<td>ButtonDownFcn</td>
<td>axes, figure, button group, panel, uiobjects</td>
</tr>
<tr>
<td>ClickedCallback</td>
<td>push tool, toggle tool</td>
</tr>
<tr>
<td>CreateFcn, DeleteFcn</td>
<td>axes, button group, context menu, figure, menu, panel, uiobjects, …</td>
</tr>
<tr>
<td>OffCallback, OnCallback</td>
<td>toggle tool</td>
</tr>
<tr>
<td>SizeChangedFcn (&gt;=R2014b)</td>
<td>figure, panel, button group</td>
</tr>
<tr>
<td>SelectionChangeFcn</td>
<td>button group</td>
</tr>
<tr>
<td>KeyPressFcn</td>
<td>figure, uiobjects</td>
</tr>
<tr>
<td>KeyReleaseFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowButtonDownFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowButtonMotionFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowButtonUpFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowKeyPressFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowKeyReleaseFcn</td>
<td>figure</td>
</tr>
<tr>
<td>WindowScrollWheelFcn</td>
<td>figure</td>
</tr>
<tr>
<td>CloseRequestFcn</td>
<td>figure</td>
</tr>
</tbody>
</table>
Functions $\text{gcf}$, $\text(gca}$ and $\text{gco}$

- serve to easily access identifiers of objects that are currently active, in particular:
  - $\text{gcf}$ – returns identifier of current object figure
  - $\text(gca}$ – returns identifier of current object axes
  - $\text{gco}$ – returns identifier of the object that was last to mouse-click on (tolerance is 5 px)

```matlab
figure
figRef = gcf
```

- these functions can be used as input identifiers for other functions requiring reference to object figure or axes

```matlab
set(gcf,'color',[0 0 0])
```
Exercise – button callback

- create figure with button and text box
- when clicking on button background color of button changes to random and displays individual RGB components in text box
Exercise – button, solution
Exercise – mouse position

- create a text array showing mouse position over figure.
- figure's callback for mouse movement is `WindowButtonMotionFcn`
- mouse position can be found in figure property `CurrentPoint`
Function findobj

- finds an object(s) with required property
- returns reference to the object (or an array of references)

>> figHndl = gcf;  \% figHndl = figure;
>> axsHndl = gca;  \% axsHndl = figure;
>> hTx1 = uicontrol('Style','text','String','hello','Tag','tx1');
>> hTx2 = uicontrol('Style','text','String','test1','Tag','tx2');

>> h = findobj('Style','text','-and','Tag','tx1')

h =

   UIControl (tx1) with properties:
       Style: 'text'
       String: 'hello'
    BackgroundColor: [0.9400 0.9400 0.9400]
    Callback: '
       Value: 0
    Position: [20 20 60 20]
       Units: 'pixels'

Show all properties

>> h = findobj('Style','text')

h =

   2x1 UIControl array:
       UIControl (tx2)
       UIControl (tx1)
Exercise – keyboard scan

- create a text array that displays last key pressed
  
  - information on the key pressed is to be found in `event` parameter
  
  - figure's callback for pressing key is `WindowKeyPressFcn`
  
  - get the reference to the text array using `findobj`
**Function** `findall, allchild`

- `findall` finds all graphic objects (including hidden)
- `allchild` finds all children of selected object (including hidden)
  - `handle_list` can be for instance `gcf`, `gca`, ...
  - if `handle_list` is a identifier vector, Matlab returns cell array

```matlab
clc, clear, close all
% figure with menu
hFig = figure;
% compare
hFig.Children
get(hFig, 'Children')
findobj('Parent', hFig)
allchild(hFig)
findall(hFig, 'Parent', hFig)
findall(hFig)
```
```matlab
clc, clear, close all
% figure with menu
hFig = figure('MenuBar', 'none');
% compare
hFig.Children
get(hFig, 'Children')
findobj('Parent', hFig)
allchild(hFig)
findall(hFig, 'Parent', hFig)
findall(hFig)
```
Function \texttt{copyobj}

- this function enables to have an influence on lifecycle of an object
  - copies object and its descendants

- more >> \texttt{doc copyobj}

\begin{verbatim}
>> hf = figure
>> ha = axes
>> hL1 = line([0.1 0.8], [0.5 0.5])
>> hL2 = copyobj(hL1,ha)
>> set(hL2, 'YData', [0.4 0.6])
>> ishandle(hL1) && ishandle(hL2)

ans =

   1
\end{verbatim}
Function delete, reset

- these functions enable to have an influence on lifecycle of an object
- delete removes file(s) or graphic object(s) together with its descendants

```
>> delete(hf) % hf see previous example
>> ishandle(hL1) && ishandle(hL2)
ans =
    0
```

- reset sets all values of an object back to implicit values

```
reset(h)
```
Advanced visualizing in Matlab

- function *gobjects* predefines variables

- function *isgraphics()*

```matlab
x = 1:10; y = sin(x);
p = plot(x,y);
ax = gca;

isgraphics([p, ax])
```

- function *ishandle* finds out whether variable is a handle object

```matlab
>> figHandle = figure;
>> ishandle(figHandle)
```

- `>> doc Graphics Object Identification`
Storing data in GUI

- how to store data in GUI?
  - global variables (extreme case, keyword `global`)
    - unacceptable
  - using property `UserData` (depends on size of the application)
    - acceptable
  - using functions `guidata` or `setappdata` and `getappdata`
    - suitable
  - fully OOP access (including functional part of the code)
    - ideal
**Function** guidata

- enables to store or get data

- the procedure is as follows:
  - get data copy: \( \text{data} = \text{guidata}(\text{object\_handle}) \)
  - carry out data modification / calculation required
  - if the data is changed, store \( \text{guidata}(\text{object\_handle}, \text{data}) \)

- data is therefore related to a handle that exist during whole lifetime of GUI
  - data is saved in object's parent figure
Function `guidata`

```matlab
>> hFig = figure('Toolbar', 'none');
>> allFigHndl = guihandles(hFig);
>> guidata(hFig, allFigHndl);
```

function `guihandles` returns references of all visible objects in figure

```matlab
function myCallback()

% ...
myAllFigHndl = guidata(gcbo);
myAllFigHndl.time = clock;
guidata(gcbo, myAllFigHndl);
```

function `gcbo` returns reference of the object callback of which is being evaluated
Functions \textit{setappdata, getappdata}

- \textbf{setappdata}: enables to define new data (pair name-value) for given application

\begin{verbatim}
clc, clear, close all

hFig = figure;
hButt = uicontrol('Parent', hFig);
setappdata(hButt, 'speedA', rand(1, 10));
\end{verbatim}

- \textbf{getappdata}: enables to get previously defined data of selected object

\begin{verbatim}
value = getappdata(hFig, 'speedA')

% values is a structure
values = getappdata(hndl)
\end{verbatim}
Exercise – mouse movements + buttons

- create application according to picture below
  - button „End“ terminates application
    - Callback of uicontrol
  - left and right mouse button click on figure changes font type of label „X-position“ and „Y-position“ from normal to bold and vice versa
    - WindowButtonDownFcn of figure and event input
  - in the case checkbox is ticked, program displays cursor position
    - Value of uicontrol and CurrentPoint of figure
Exercise – mouse movements + buttons
Exercise – mouse movements + buttons
Predefined dialog windows

- The most common operations used ↔ GUI are predefined
- the most common ones are displayed below (most of them):

<table>
<thead>
<tr>
<th>→ user</th>
<th>helpdlg</th>
<th>msgbox</th>
<th>warndlg</th>
<th>errordlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ GUI</td>
<td>inputdlg</td>
<td>listdlg</td>
<td>questdlg</td>
<td></td>
</tr>
<tr>
<td>file →</td>
<td>uigetdir</td>
<td>uigetfile</td>
<td>uiopen</td>
<td></td>
</tr>
<tr>
<td>→ file</td>
<td>uiputfile</td>
<td>uisave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>→ user</td>
<td>waitbar</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Function** `msgbox`

- displays message for the user

```matlab
>> h = msgbox({'This is a message for Matlab students.',... 'Predefined functions save time',... 'Demonstration of msgbox usage.'})
```

![Message box example](image)
Function `questdlg`

- displays a question, returns answer

```matlab
>> query = questdlg('Terminate application?', 'End of application', 'Yes', 'No', 'Yes')
```

![Screenshot of the `questdlg` function dialog box](image)
Function `uigetfile`

- user can select file(s) from file dialog box
- files can be filtered by their suffix

```matlab
>> [FileName, PathName] = uigetfile('*.m', 'Select the M-file');
```
Function **uiputfile**

- opens dialog for file saving
- files can be filtered by their suffix

```
>> [file,path] = uiputfile('.mat', 'Save workspace as:', ...
             'defaultFile.mat')
```
Exercise – saving into file

- save variable data from Workspace in a file using dialog box
Function \texttt{waitbar}

- displays state of a process

```matlab
h = waitbar(0, 'Please wait...');
nsteps = 1000;
for k = 1:nsteps
    waitbar(k/nsteps);
end
close(h);
```
Design of a simple GUI #1

- what the GUI should do (detailed description of functionality)
- what are the user inputs
- required outputs

- objects used (scheme of GUI, list of elements, design of tags and properties)
- callback functions, dynamic elements

- saving of identifiers and data in GUI
- programming style

- implementation of individual parts

- getting it to work, testing...
Discussed functions

gcf, gca, gco
findobj, findall, allchild
copyobj
delete, reset
gobjects, ishandle, isgraphics
helpdlg, msgbox, warndlg, errordlg
inputdlg, listdlg, questdlg
uigetdir, uigetfile, uiopen
uiputfile, uisave
waitbar
guide
guidata, setappdata, getappdata
Exercise – displaying graph of a function

- expand previous function so that it enabled to draw graph of a function defined by user
  - use `try—catch` to eliminate erroneous inputs
  - use function `reset` to clear graph before another drawing
  - what function do you use to evaluate the text input?

```
>> MTB_GUI1edit
```

![Graph of a function](attachment:image.png)
Exercise – displaying graph of a function
Exercise – displaying graph of a function
Thank you!

Apart from educational purposes at CTU, this document may be reproduced, stored or transmitted only with the prior permission of the authors. Document created as part of A0B17MTB course.

ver. 6.2 (20/12/2016)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz