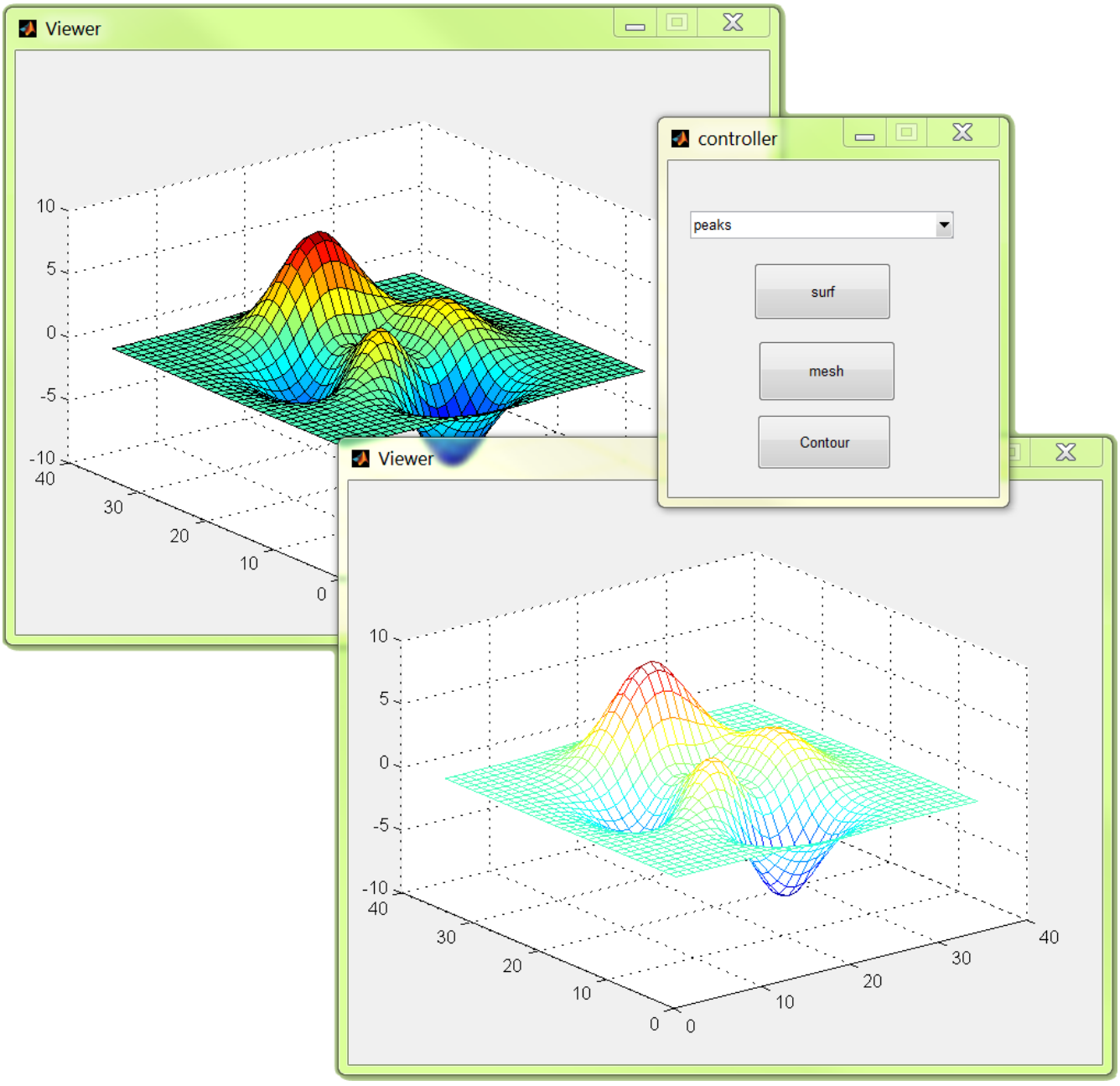
Create a Simple Object Oriented GUIDE GUI in MatLAB



Esben Jannik Bjerrum

This example shows how to create a simple graphical user interface (GUI), using GUIDE and a Model-Controller-Viewer like organization (Figure 1). The example draws on the Simple GUI tutorial from the Mathworks documentation: http://www.mathworks.se/help/matlab/creating_guis/about-the-simple-guide-gui-example.html, If unfamiliar with Guide and GUI developement, follow that tutorial first.
The structure of the program is however different from the MathWorks tutorial, and for such a simple project as here, it may be overkill. However, the program will get easier to expand and troubleshoot, as the individual GUI elements get much more independent of each other. setdata/getdata pairs to pass data back and forth between different GUI's are avoided. Methods, properties and data manipulation routines are kept in a single place. The central data object can be used without the GUI and are thus easier to test, debug and script directly from the command window. The properties have a defined name, and misspelling in a set elsewhere will result in an error. Read more about the concept and pros and cons at http://en.wikipedia.org/wiki/Model-view-controller

The model is coded as a class, where the user manipulates the object through the controller, and the viewer watches it. Read more about the concept at http://en.wikipedia.org/wiki/Model-view-controller

Code the Model class

Start simple, open a new file in matlab and enter the following:

```
classdef Data < handle
    %Data represents a simple object as a
data container with only one property
    properties (SetObservable = true)
        current_data % Current Data to plot
    end
end
```

Save the file as Data.m. This acts as the model and holds the data.

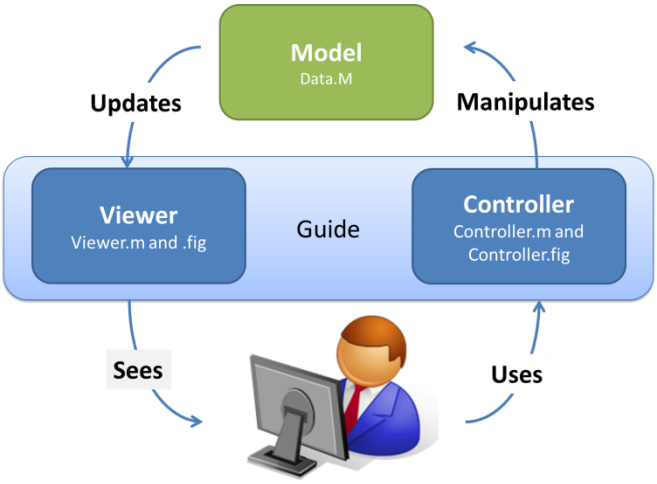


Figure 1 Schematical representation of the MCV architecture.

Test the object in the MatLAB command window:

```
>> d = Data

d =

  Data with properties:

    current_data: []
```

Expand the object to get more functionality. Add some private, hidden properties below the other properties and a method section with a function that has the same name as the class. The function will be executed upon initialization of the object.

```
properties (SetAccess = private, Hidden = true)
        peaks %Precomputed peaks data
        membrane %Precomputed membrane data
        sinc %Precomputed sinc data
end
```

```
methods
        function obj = Data(varargin) %Initialise the object
            obj.peaks=peaks(35);
            obj.membrane=membrane;
            [x,y] = meshgrid(-8:.5:8);
            r = sqrt(x.^2+y.^2) + eps;
            sinc = sin(r)./r;
            obj.sinc = sinc;
        end
end
```

But this only fills some hidden, private properties. To control which data are the current, add a get.function to the methods section. This function will run whenever the Data.current_data are called, and will return the value assigned to temp variable data by the function.

```
        function data = get.current_data(obj)
            %This code runs upon access of current_data
            switch obj.selected_data
                case 'peaks'
                    data = obj.peaks;
                case 'membrane'
                    data = obj.membrane;
                case 'sinc'
                    data = obj.sinc;
            end
        end
```

Additionally, add a property to control the selection of the data in the public property clause.

```
selected_data = 'peaks'
```

Try and initialise the data object and set the selected_data. The current_data will change.

```
>> clear
>> d = Data

d =

  Data with properties:

    current_data: [35x35 double]
   selected_data: 'peaks'

>> d.selected_data = 'membrane'

d =

  Data with properties:
```

current_data: [31x31 double]
selected_data: 'membrane'

Lastly, add two events (one needed later in the GUI), and a set.function in the methods section to control if the selected_data property gets a proper value.

In a section just below the properties

```
    events
        dataChanged % The exposed data has changed
        selecterror % An error occurred
    end
```

and in the methods section

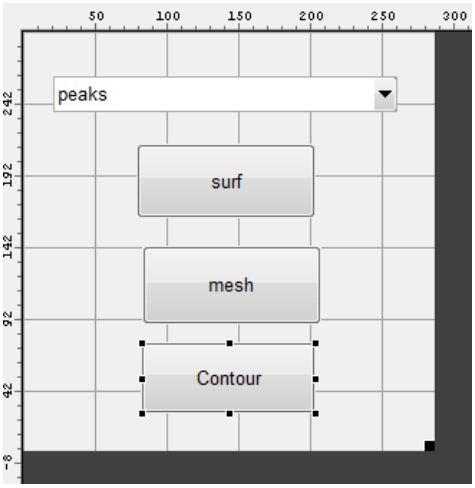
```
        function set.selected_data(obj, selection)
            if ismember(selection, ['peaks' 'membrane' 'sinc'])
                obj.selected_data = selection;
                notify(obj,'dataChanged'); %Notify event (and anything listening),
that the selected data has changed
            else
                notify(obj,'selecterror')% Notify that an error has occured
                display('Selected data must be ''peaks'', ''membrane'' or
''sinc'''); %Print to command window
            end
        end
```

The set.function, gets executed whenever the data.selected_data are assigned a new value. It will assign the supplied value to the property and notify the event, or issue an error message.

The final Data.m can be found in Appendix 1.

Make the Controller in GUIDE:

Open a New GUI in the GUIDE Layout Editor, start GUIDE by typing `guide` at the MATLAB prompt. In the GUIDE Quick Start dialog box, select the **Blank GUI (Default)** template, and then click **OK**.



Populate the GUI with a popup menu and three buttons. Edit the string property of the popupmenu to contain peaks, membrane and sinc on separate lines. Rename the buttons and assign their tag's to surf, membrane and contour, respectively.

Save as e.g. Controller.m and .fig in the same directory as the Data.m

To code the GUI, first get the Controller to start a Data object upon initialization. Add the following code to the OpeningFcn of the Controller.m file, just before the guidata statement. This way a handle for the Data object are added to the GUI's

handles object, which enable easy access in the other functions and callbacks of the GUI.

```
%Set the model
handles.data = Data();
```

Find the call back from the popupmenu, and make it manipulate the Data.selected_data. The following code extracts the content list of the popupmenu, and uses the value of the popupmenu to get the selected menu item as a string. Handles.data.selected data then gets this string. Changes to the Data object are instant. No need to run a guidata(hObject, handles).

```
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
contents = get(hObject,'string');
handles.data.selected_data = cell2str(contents(get(hObject,'value')));
```

For the buttons which should launch the viewer, make a common function.

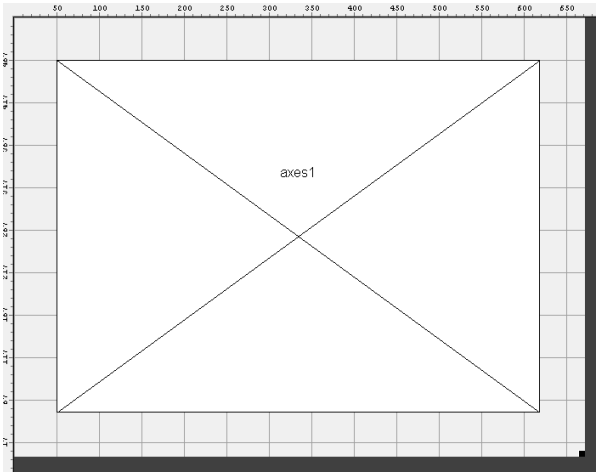
```
function Button_Callback(hObject, eventdata, handles)
          tag = get(hObject,'Tag');
          Viewer(handles.data, tag);
```

In the guide GUI editor, set the callback function of all buttons to @(hObject,eventdata)controller('**Button**_Callback',hObject,eventdata,guidata(hObject))

Basically, just change surface, mesh or contour to Button. Save the GUI fig. The button callback will start a Viewer, and pass it a handle to the Data object and the tag of the button which launched it. In all it was 6 lines of code, apart from the GUIDE generated code. The final code including all the GUIDE generated stuff can be found in Appendix 2.

Make the Viewer in Guide.

Open a new blank guide gui. Place an axes object on it. Click Tools, GUI options and deselect “GUI allows only one instance to run”. Save it as Viewer.m (and .fig).



Edit the code to accept the handle to the Data object and the tag, by putting these two lines in the opening function.

```
%Decipher the varargin (Expect model and
tag)
handles.data = varargin{1};
handles.buttontag = varargin{2};
```

Add a function to plot to the axes object, depending on the tag.

```
function onChangedData(handles, data)
%Depending on tag fed to GUI, select how to plot
switch handles.buttontag;
    case 'surf'
        surf(data.current_data,'parent',handles.axes1);%'parent',handles.axes1
prevent window from stealing focus.
    case 'mesh'
        mesh(data.current_data,'parent',handles.axes1);
    case 'contour'
        contour(data.current_data,'parent',handles.axes1);
end
```

to make the viewer aware of what is happening with the Data object, add a listener to the OpeningFcn (before guidata(hObject, handles).

```
%Listen for change event
handles.listen = event.listener(handles.data, 'dataChanged', @(o,e)
onChangedData(handles,handles.data));
```

The listener is triggered, when the 'dataChanged' event is notified, and in turns run the onChangedData function. (o,e) are the triggering object and the event data, which are not used here. For persistency it is added to the handles struct, which is saved with the GUI later in the Opening_Fcn.

Finally, add a call to onChangedData once as initialisation of the GUI. Put in OpeningFcn, AFTER the guidata command.

```
onChangedData(handles,handles.data);
```

The final Opening_fcn and onDataChanged can be seen in Appendix 3. Apart from the guide generated code, it was 13 lines of code.

Run the GUI by from the controller file. Open a couple of viewer windows, and see what happens if the data selection is changed by the controller window.

The error notification event was never utilised in the GUI elements.

Additional notes on MCV in MatLAB

When working with plots, always be sure to specify which axes element are plotted to. The current figure will change depending on which GUI is in focus, and plotting commands that plots to the current figure may draw the plot in unexpected places.

If the user clicks wildly around, launching new Viewer windows in quick succession, race conditions and unexpected behaviour may result.

The Create_fcn for the GUI elements of the guide generated GUI's run before the Opening_fcn. Thus the create functions will not have access to the model object, which get added to the handles structure in the Opening_fcn. Population of e.g. popup menus must thus be added to the Opening_Fcn or in a function that gets called from the Opening_Fcn.

'membrane', 'peaks' and 'sinc', have been added explicitly in the code as strings in a lot of places for illustrative purposes. However, it may be a good idea to add the list of selectable strings directly to the object, and then use this list to populate the popup and select the current_data based on a passed index, instead of string matching.

It is not necessary to split the viewer and controller in separate windows, as this may give a very confusing and cluttered experience with a lot of windows. The viewer and controller parts can just as well be built together in the same guide GUI.

Writing comments in the correct place in the Data.m file, will enable help and doc commands. e.g. >>help Data and >>help Data.selected_data. The doc command requires that the Data.m file is on the matlab path.

Hope it will be useful to someone.

Best Regards

Esben Jannik Bjerrum

www.wildcardconsulting.dk

Appendix1: The final Data.m

```
classdef Data < handle
    %Data represents a simple object as a data container with only one
    %property exposing precomputed data depending on a setting.
    properties (SetObservable = true)
        current_data  % Current Data to plot

        % Set the current data to report
        % Valid options are 'peaks', 'membrane' or 'sinc'
        selected_data = 'peaks' %The above comments are exposed shown doc.
    end
    properties (SetAccess = private, Hidden = true)
        peaks %Precomputed peaks data
        membrane %Precomputed membrane data
        sinc %Precomputed sinc data
    end
    events
        dataChanged % The exposed data has changed
        selecterror % An error occurred
    end
    methods
        function obj = Data(varargin) %Initialise the object
            obj.peaks=peaks(35);
            obj.membrane=membrane;
            [x,y] = meshgrid(-8:.5:8);
            r = sqrt(x.^2+y.^2) + eps;
            sinc = sin(r)./r;
            obj.sinc = sinc;
        end
        function data = get.current_data(obj)
            %This code runs upon access of property
            switch obj.selected_data
                case 'peaks'
                    data = obj.peaks;
                case 'membrane'
                    data = obj.membrane;
                case 'sinc'
                    data = obj.sinc;
            end
        end
        function set.selected_data(obj, selection)
            if ismember(selection, ['peaks' 'membrane' 'sinc'])
                obj.selected_data = selection;
                notify(obj,'dataChanged'); %Notify event (and anything listening),
that the selected data has changed
            else
                notify(obj,'selecterror')% Notify that an error has occured
                errordlg('Selected data must be ''peaks'', ''membrane'' or
''sinc'''); %Print to command window
            end
        end
    end %method
```

```
end %Object
```

Appendix 2: The final Controller.m

Bold are non-GUIDE generated additions.

```
function varargout = controller(varargin)
% CONTROLLER MATLAB code for controller.fig
%      CONTROLLER, by itself, creates a new CONTROLLER or raises the existing
%      singleton*.
%
%      H = CONTROLLER returns the handle to a new CONTROLLER or the handle to
%      the existing singleton*.
%
%      CONTROLLER('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CONTROLLER.M with the given input arguments.
%
%      CONTROLLER('Property','Value',...) creates a new CONTROLLER or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before controller_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to controller_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help controller

% Last Modified by GUIDE v2.5 30-Sep-2014 10:07:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @controller_OpeningFcn, ...
                   'gui_OutputFcn',  @controller_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 controller is made visible.
function controller_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 controller (see VARARGIN)

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

%Initialise the Data object and add it to the handles struct
handles.data = Data();

% Update handles structure
guidata(hObject, handles);

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

function Button_Callback(hObject, eventdata, handles)
% Launch a viewer GUI, passing a handle for the data and the tag of the calling
button
tag = get(hObject,'Tag')
Viewer(handles.data, tag)


% --- Outputs from this function are returned to the command line.
function varargout = controller_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 selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
contents = get(hObject,'string');
handles.data.selected_data = cell2str(contents(get(hObject,'value')));
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
```

Appendix 3: The final Viewer.m

Bold are non-GUIDE generated additions.

```
function varargout = Viewer(varargin)
% VIEWER MATLAB code for Viewer.fig
%      VIEWER, by itself, creates a new VIEWER or raises the existing
%      singleton*.
%
%      H = VIEWER returns the handle to a new VIEWER or the handle to
%      the existing singleton*.
%
%      VIEWER('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in VIEWER.M with the given input arguments.
%
%      VIEWER('Property','Value',...) creates a new VIEWER or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Viewer_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Viewer_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Viewer

% Last Modified by GUIDE v2.5 30-Sep-2014 10:34:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 0;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Viewer_OpeningFcn, ...
                   'gui_OutputFcn',  @Viewer_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 Viewer is made visible.
function Viewer_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 Viewer (see VARARGIN)

%Decipher the varargin (Expect model and tag)
handles.data = varargin{1};
handles.buttontag = varargin{2};

%Listen for change event
handles.listen = event.listener(handles.data, 'dataChanged', @(o,e)
onChangedData(handles,handles.data));

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

% Update handles structure
guidata(hObject, handles);
%Run onChangedData to initalise the new figure.
onChangedData(handles,handles.data);
% UIWAIT makes Viewer wait for user response (see UIRESUME)
% uiwait(handles.figure1);

function onChangedData(handles)
%Depending on tag fed to GUI, select how to plot
switch handles.buttontag;
    case 'surf'

surf(handles.data.current_data,'parent',handles.axes1);%'parent',handles.axes1
prevent window from stealing focus.
    case 'mesh'
        mesh(handles.data.current_data,'parent',handles.axes1);
    case 'contour'
        contour(handles.data.current_data,'parent',handles.axes1);
end


% --- Outputs from this function are returned to the command line.
function varargout = Viewer_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;
```