ADVANCED MATLAB GUI DEVELOPMENT WITH THE DataGUI LIBRARY

Sascha Meykopff, DESY, Hamburg, Germany

Abstract

On the DESY campus Matlab is a widely used tool for creating complex user interfaces. Although the on-board GUI tools are easy to use and provide quick results, the generated low-level code lacks uniformity and advanced features like automatic verification and conversion of input and output data. These limitations are overcome by the newly developed DataGUI library. The library is based on the model-view-controller software pattern and supports enhanced data handling, undocumented Matlab GUI elements, and configurable resizing of the user interface. An outlook on features of the upcoming release is also presented.

INTRODUCTION

At the European XFEL and the FLASH facility, both located on the DESY campus, most of the operation handling is based on the client-server model with JDDD as the front-end client and DOOCS servers as the back end software [1]. If an operation task needs a more complex graphical user interface (GUI) it's recommended to use Matlab. The GUIDE tool is an interactive method to create a GUI with Matlab. The use of GUIDE provides quick results but extensive code work is necessary to handle the interaction between the GUI elements and the program logic. The effort to implement a GUI complete programmatically without the help of GUIDE is small. The DataGUI library was developed to reduce the coding amount, and to improve the stability of Matlab applications.

ACCESS AND VERIFY PROPERTIES

Matlab GUI elements are designed as objects. The GUI element behavior is defined by object properties. To access element properties the developer has to call a get or set function. Starting with Matlab version R2014b one can use the dot notation to query or set properties [2]. The property names and type format depend on the GUI element type and Matlab version. To write an application which works on different Matlab releases these circumstances must be considered.

```
function edit_field_callback(handle, event)
   edit_input = get(handle, 'STRING');
   edit_input = str2double(edit_input);
   if ~isnan(edit_input)
      display(['input is ' num2str(edit_input)])
   else
      display('input invalid')
   end
```

Figure 1: How to verify and convert text field input in a Matlab callback function.

In a good software design every input variable should be verified. To neglect parameter verification results in unstable software. Currently a Matlab developer is not encouraged to verify the GUI input. Figure 1 shows an example how to verify and convert an edit text field for integer input. This code has to be implemented for every input element.

SHARE DATA AMONG CALLBACKS

The interaction between GUI elements and user code is done by callback functions. If one clicks on a GUI button, Matlab calls an assigned function. Most of the Matlab callbacks carry two input and no output parameters. The first parameter is the handle of the triggered GUI element. The second parameter is defined as 'eventdata' without any value. The developer can add additional input parameters. No return parameters are defined in Matlab callbacks. If the code needs to share data among callbacks the function parameters are not suitable. The Matlab documentation describes four different solutions [3]. These solutions have in common to use a function to get and store the shared data. These functions needs to be called at the start and the end of every callback. This design causes a lot of code duplication and potential errors. A lot of real world software use global variables to store data among callbacks. This is one of the worst solutions because global variables yields in a bad code design.

MODEL-VIEW-CONTROLLER

The Model-view-controller (MVC) is a software pattern which is followed by the most modern GUI toolkits. The pattern is an abstract idea of separating the code into three parts. One part describes and handles the data model of an application. The second one creates the view of the data model. Multiple views of the same data model are allowed for example to view a data table and a plot. The last part covers the program logic. This part updates the data model and modifies the different views. The MVC pattern gives a rough software structure and helps the developer to create a good software design. The software design usually suffers because developing a Matlab GUI is time-consuming. The DataGUI library guides the developer to separate his code which results in a better design.

GLOBAL DATA DESCRIPTION

Matlab software who uses the DataGUI library has unified callbacks. Every callback has a 'data' variable as input and output. The type of the 'data' variable is a structure. One can add, remove, and store fields inside the structure. The storage among callbacks is handled by the

ISBN 978-3-95450-148-9

DataGUI library. The library defines special fields in the data structure called 'registered variables'. These registered variables have additional properties. The library stores the type of the variable. An automatic data conversion ensures the variable type if this variable is modified by a GUI input element. Additionally the library supports constraints. For example a numeric variable has a minimal and a maximal value. The constraints is checked if an input element changes the registered variable. If the input value violates the constraints no modification of the data structure is happen and the invalid input element is marked red. There is no invalid data values in the registered variables. Another additional parameter of a registered variable is a callback definition. If a GUI element has modified a registered variable this function is called. In DataGUI code the callbacks are bind to registered variables not to GUI elements. If different GUI elements modifies the same registered variable the same callback is executed. Figure 2 shows an example for a variable definitions. The definition of the registered variables conform to the view part of the MVC pattern.

```
i = cell( {
   {'data.input_string' 'Test string'              }
   {'data.input_num'    1           [0 10] @new_num}
   {'data.turnoff'      'on'                       }
});
data = data.CALL.addData(data, i);
```

Figure 2: Example code of data definition. Registered variable 'input_string' with value 'Test string'. Variable 'input_num' as numeric input in the range from 0 to 10 and a callback definition 'new_num'. The addData function creates uses the cell array as input and creates all variables.

GUI ELEMENT DEFINITION

The DataGUI library handles the creation of new GUI elements. A new GUI element is defined by the type string of the new element, the new tag name, and the parent tag name. Depending on the type of the new element some parameters are necessary. For example a new label element needs a string as parameter. This string defines the displayed output. If this parameter value is the name of a registered variable the behavior is different. The output of this GUI element is now linked to the registered variable and the element displays it's current value. If the value of the variable is modified inside callback function the GUI element will be changed direct after the callback is finished. If the new GUI element is an edit field the mandatory parameter should be linked to a registered variable. If a user changes the value of the edit field the library read the current value of the GUI element. In the next step the value is converted to expected type format and the constraints is verified. If this verification is successful the new value is stored in the registered variable and the associated callback function is executed. The developer modifies GUI elements by assigning new values to the registered variables. No exception handling because of invalid input parameter is necessary. This will decrease the code size and improves the software quality. In most cases the proper definition of new GUI elements require more parameters. Additional parameters can be defined by property/value pairs. For example the 'visible' property of a GUI element can be set to the value 'off'. But this value also can be the name of a registered variable. In this case the property status depends on the value of the registered variable. A developer can change the visibility status by writing 'on' or 'off' into the linked variable. A small example about the GUI element definition with DataGUI is shown in Figure 3 and the resulting GUI in Figure 4.

```
g = cell( {
    {'figure'   'fig1'   'Name' 'Constant title string'                 }
    {'edittext' 'edit1'  'fig1' 'data.input_string'                      }
    {'edittext' 'edit2'  'fig1' 'data.input_num'                         }
    {'edittext' 'edit3'  'fig1' 'data.input_num' 'visible' 'data.turnoff'}
});
data = data.CALL.addGui(data, g);
```

Figure 3: Defining a GUI with 4 elements. 'Name' and 'visible' are additional parameters.

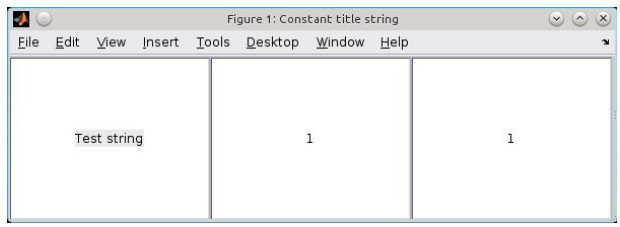

Figure 4: Example GUI with code of Figures 2,3,6.

ADDITIONAL FEATURES OF DATAGUI

The DataGUI library supports all standard Matlab GUI elements. Additionally support for undocumented GUI elements is implemented.

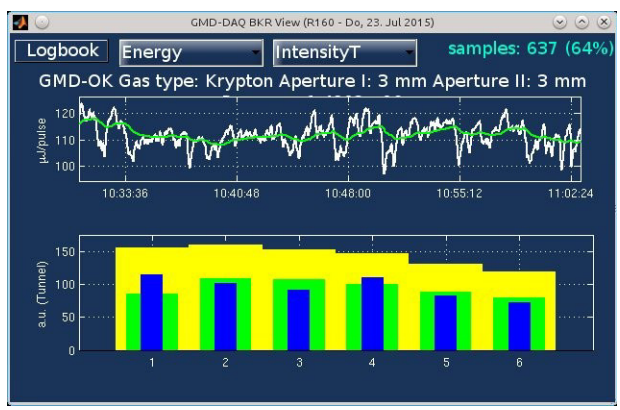

Figure 5: Example of advanced event support. The upper plot can be scale by mouse drag and drop.

ISBN 978-3-95450-148-9

This covers tabulator support for older Matlab versions with a consistent interface. Tree elements with a convenient data interface are included. Dynamic trees obtain their data by a callback function. Also timers are handled by the library. Matlab timers are difficult to handle. The callback function of a timer is able to interrupt a GUI callback. If the timer interacts with the GUI or other data this behavior result in unstable conditions. The timer of the DataGUI library will not run a callback if another callback is executed. A special timer is implemented which repeatedly executes a callback function while a defined condition is true. For example this could be used to emulate multi-threading functionality. A unified callback interface for undocumented callbacks supports the developer to implement a drag and drop interface or to catch other advanced mouse events. An application which uses mouse events to implement a drag and drop control of a plot is shown in Figure 5.

RESIZING OF A MATLAB GUI

The Matlab GUI supports the resizing of elements. A GUI created by GUIDE resizes every element relative to the figure size or not at all. On modern computer systems with huge visible monitor sizes this behavior is unsuitable. For example axes should be resized in a different way than edit fields. To handle the resize of a figure a developer has to set the 'ResizeFcn' or in newer Matlab versions the 'SizeChangedFcn' property. Inside the referred callback function the developer has to calculate and set the complete layout of all visible elements. The DataGUI library shorten this work.

```
layout = {{0 0 0} {'edit1' 'edit2' 'edit3'}};
data = data.CALL.addLayout(data, 'fig1', layout);
```

Figure 6: Example layout definition.

The layout of a GUI is defined by a nested cell array. Each cell describes a horizontal or vertical layout. The cell contains pairs of targets and size definitions. The target is the tag names of GUI element or another nested cell array. The size definition is the size in pixels, a fraction, or the remaining space of the layout. It's possible to use registered variables as a size parameter. This allows the developer to change the layout dynamically. An example layout definition is in Figure 6. Figures 7,8 show a more complex example for a nested structure.

```
h_lay = {{ 0 0 } {'edit1' 'edit2'}};
v_lay = {{ 30 30 };{'edit3' h_lay}};
data = data.CALL.addLayout(data, 'fig1', v_lay);
```

Figure 7: Nested cell array example.

OUTLOOK

While the development of control room applications we improved the library a lot (see Figure 9). But some topics have to be done.

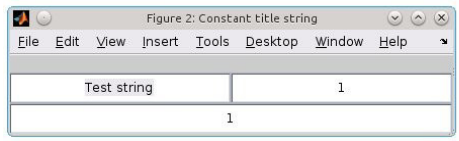

Figure 8: Result of code shown in Figure 7.

One topic is a new object oriented design of the layout to improve the legibility of the code. With the release 2014b of Matlab some fundamental changes happened. This results in a different timing behavior of the applications and the support for special mouse events must be revised. In discussions with different developers a general object oriented approach of the DataGUI library can be considered.

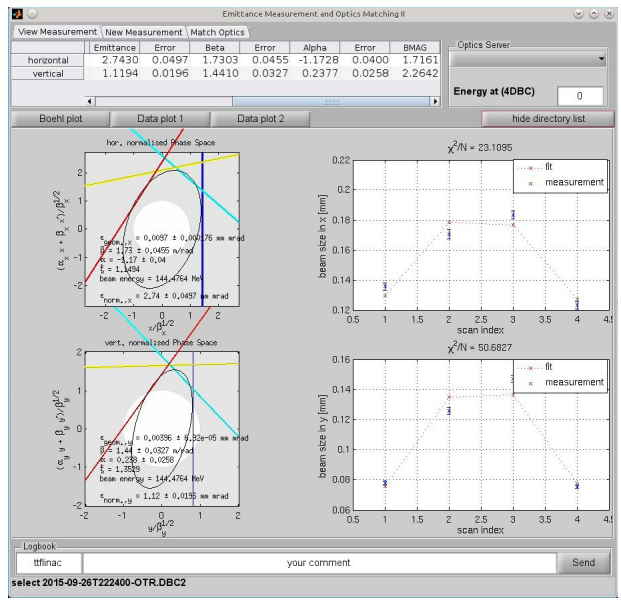

Figure 9: The new emittance measurement and match tool uses the DataGUI library.

CONCLUSION

The DataGUI library enables to build full featured graphical user interfaces with Matlab. The design of the API separates data handing and GUI element handling which will result in a much cleaner code structure. To reduce code duplications and to improve code stability the library takes care about converting parameters and observes constraints. The library supports a wide array of modern GUI elements. The different Matlab API over a long range of Matlab releases is considerd. The DataGUI library supports extensive possibilities to resize the layout. As demonstrated in the DESY control-room the library offers powerful, stable, and easy to handle Matlab software suitable for the daily use by the machine operators.

ISBN 978-3-95450-148-9

REFERENCES

[1] R. Bacher et al., "The large scale European XFEL control system: Overview and status of the commissioning", MOA3O02, Proc. ICALEPCS'15, Melbourne, Australia, (2015)

[2] Matlab release notes over a range of versions: https://de.mathworks.com/help/matlab/release-notes.html/.

[3] Matlab manual (share data among callbacks): https://de.mathworks.com/help/matlab/creating_guis/share-data-among-callbacks.html/.

ISBN 978-3-95450-148-9