New ROOT graphics language

Iliana Betsou (1), Serguei Linev (2), Bertrand Bellenot (1), Olivier Couet (1).

(1) CERN: European Laboratory for Particles Physics - Esplanade. des Particules 1, 1211 Meyrin - Switzerland
(2) GSI Helmholtzzentrum für Schwerionenforschung GmbH - Planckstraße 1 64291 Darmstadt - Germany

E-mail: iliana.betsou@cern.ch, s.linev@gsi.de, bertrand.bellenot@cern.ch, olivier.couet@cern.ch

Abstract. In the context of ROOT7, the graphics system is completely redefined. Based on client server architecture and with the use of modern C++ and JavaScript, ROOT7 provides a new web based graphics system. The new concepts of ROOT7 can be displayed directly in the browsers using the new classes for opening a new web window, communicate with the server and exchange data between front and back end and JavaScript ROOT (JSROOT).

1. Introduction

ROOT [1] exists for more than two decades and from the beginning, it has its own graphics system. This graphics system [2] used the most popular technologies at that time, like OpenGL and X11. With the rapid development of technology of the last years and the high turn on the web technology, it was time for changing the whole graphics system. The new ROOT7's graphics library in ROOT, with the use of modern C++ and web based architecture, redefine completely ROOT graphics (see Figure 1) and introduce a new web based environment.

2. Objectives of the new version, namely ROOT7

The current version of ROOT6 [3] Graphical User Interface (GUI) is very "OS-specific" and originally based on X11. Meanwhile there are different problems, including missing support of X11 on MacOS [4] or difficulties to use OpenGL through remote X11. Another limiting factor of the current implementation is the missing support of multi-threading.

The use of web platforms would improve portability, remote display support, and the adoption of future platforms, ensuring more robust maintenance. In addition, with this new model, support for multi-threading is assured. The answer to all the above can be the ROOT7 [5] development.

3. ROOT7

A web based application (as ROOT7) is a client/server computer application (Figure 2), where the server is a standard ROOT C++ application and the client runs directly in the browser. The most important classes on the C++ side are *THttpServer* [6] which is handling the communication between the different parts and *TBufferJSON* [7] which is in charge of

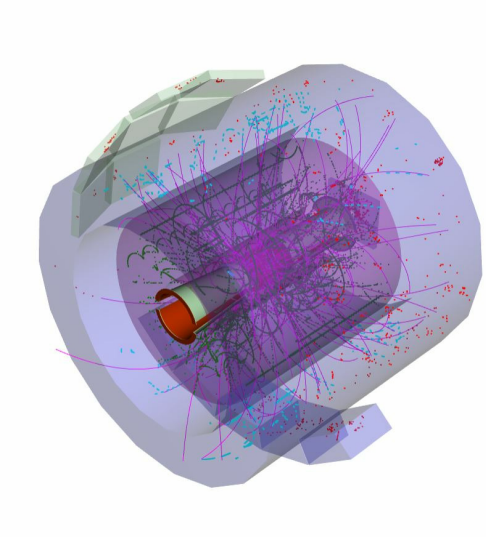

Figure 1. 3D view of the ALICE detector showing tracks and hits, using ROOT

the Input/Output (I/O). The client side is written in JavaScript and uses JSROOT [8]. For the implementation of high-level widgets, the *OpenUI5* [9] framework is used facilitating the Model-View-Controller (MVC) [10] architecture. The components are described in the following sub-sections.

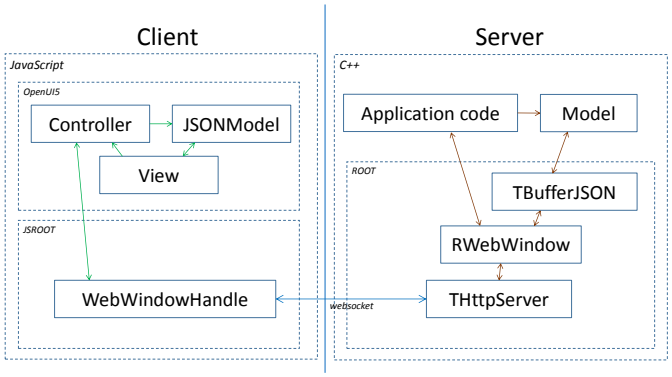

Figure 2. Client Server model: the connection between the client side and the server side is made with a Browser using a web socket

3.1. JSROOT

JSROOT is being developed since 2012 and is responsible for displaying the ROOT objects in web browsers using JavaScript. The displayed objects can be manipulated directly in the web browser. JSROOT can read binary data, stored in TFile, including *TTree*. Also the ROOT JSON format is fully supported. JSROOT is used by the ROOT Jupyter [11] interface.

3.2. THttpServer

THttpServer is one of the most important components [6]. It gives HTTP access to any running ROOT application and is responsible for the execution of commands and methods. With *THttpServer* and JSROOT, it is possible to visualize ROOT objects. The communication is achieved via *WebSockets* protocol, which is bidirectional and supports binary data. *WebSocket* is a communication protocol providing full-duplex communication channels over a single TCP connection.

3.3. TBufferJSON

TBufferJSON can convert any streamable C++ object to JSON and vice versa [7]; the custom streamers can also be supported. With the use of *TBufferJSON*, the data exchange between the server (C++) and the client (JavaScript) is simple. The real ROOT I/O remains fully on the server side.

3.4. OpenUI5

The design of the GUI layout is based on *OpenUI5* [9] which is a JavaScript UI library consisting of a really large number of UI controls. *OpenUI5* implements the MVC [10] architecture, which split the application logic into three parts:

- The Model component contains the application's dynamic data.
- The View component is the final output that ends up in the user's browser and defines how the application's data should be displayed.
- The Controller component receives inputs and contains the logic that updates the model or the view, depending on the inputs.

3.5. RWebWindow Class

The RWebWindow [12] class is a server-side entity in the new ROOT window management. It displays windows in the web browser and manages multiple connections with the clients. This class transfers the data from and to the clients. It also supports the batch mode (by using *headless* mode).

3.6. Graphics Model

The new graphics model is independent from any local graphics backend. It allows remote display on all kind of devices. The JavaScript rendering is able to be performed in a local canvas.

There are three main components in the ROOT7 graphics model:

a) The *Pad* is a base entity containing the list of graphics objects to be drawn and is implemented by the *RPad* class.

b) The *Canvas* is the window's topmost pad, implemented by *RCanvas*.

c) The *Drawable* entity. A drawable can be anything that can be drawn on a pad. Each drawable entity has a `GetDrawable` method and is implemented in the *RDrawable* class.

3.7. Batch Output

In the current state of ROOT, there are two main kinds of batch output images:

- **Vector graphics output** (like PDF and SVG) which is implemented by the native ROOT classes
- **Bitmap output** (like PNG and JPEG) which is implemented natively on top of *libAfterImage*

In the new ROOT7, the batch output doesn't rely on any dedicated ROOT graphics libraries, but it is based on the *headless mode* provided by the web browsers. In this mode, the graphics is generated in the background by the browser, re-using exactly the same code again and storing result in the output files.

4. Example: Fit Panel

The new web-based version of *Fit Panel* is an example of the use of all the above, in particular the *OpenUI5* environment. The *Fit Panel* is a simple user interface, providing all the power of the ROOT fitting tools to interactively fit histograms. One could select predefined functions (such as Gaus, Exponential, Polynomial) or any function defined by the user. There are options for different ROOT libraries and minimization methods along with a lot of fit and draw options as well as specific range selector to define the range of the histogram to be fit. The new and old layout of the *Fit Panel* is shown on Figures 3 and 4.

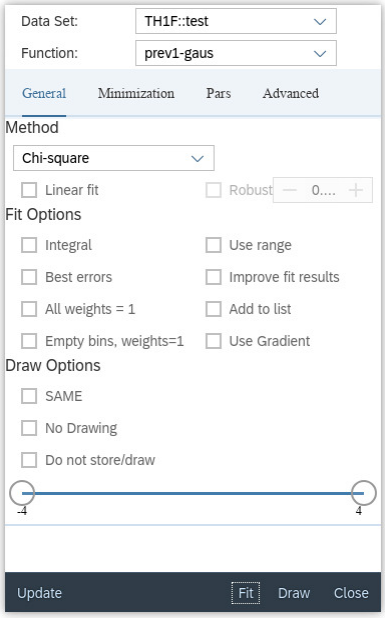

Figure 3. ROOT7 version of the Fit Panel

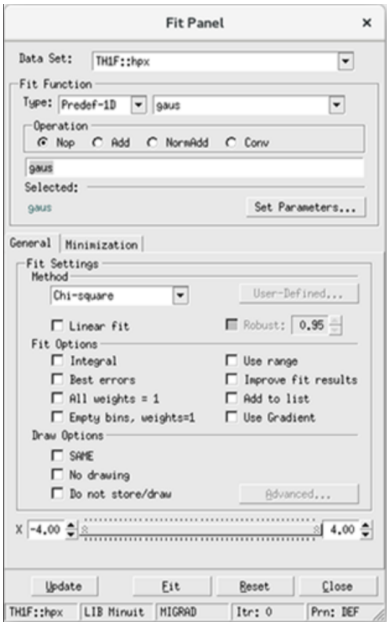

Figure 4. ROOT6 version of the Fit Panel

The *Fit Panel* includes a lot of different UI controls - text, items lists, range selectors. Values for these controls provided through *JSONModel* class of *OpenUI5*. A small but indicative example shows how the *ComboBox* control can be dynamically filled and selection results can be used on the C++ side.

The first step is the creation of C++ structures, which can represent all necessary data for *ComboBox* is described in the following code:

```
// single item
struct RComboBoxItem {
   std::string key;
   std::string value;
   RComboBoxItem() = default;
   RComboBoxItem(const std::string &k, const std::string &v) : key(k), value(v) {}
};
// complete structure
```

```
struct RFitPanelModel {
   std::vector<RComboBoxItem> fItems;
   std::string fSelected;
};
```

The model structure includes vector of items *fItems* and selected element *fSelected*. This structure should be filled with the actual values and send to the client as JSON, using the *TBufferJSON* class for transformation The C++ code below shows how to create three combo box items:

```
RFitPanelModel model;
model.fItems = {{"id0", "value0"}, {"id1", "value1"}, {"id2", "value2"}};
model.fSelected = "id1";
auto json = TBufferJSON::ToJSON(&model);
fWindow->Send(fConnId, "MODEL:"s + json.Data());
```

The C++ code above produces the following JSON output:

```
{
  "_typename" : "RFitPanelModel",
  "fItems" : [{
    "_typename" : "RComboBoxItem",
    "key" : "id0",
    "value" : "value0"
  }, {
    "_typename" : "RComboBoxItem",
    "key" : "id1",
    "value" : "value1"
  }, {
    "_typename" : "RComboBoxItem",
    "key" : "id2",
    "value" : "value2"
  }],
  "fSelected" : "id1"
}
```

When such data is received by the JavaScript client, it should be parsed and the model object assigned to the view. Here is the JavaScript code showing how this is done:

```
if (msg.find("MODEL:") == 0) {
   obj = JSROOT.parse(msg.substr(6));
   this.getView().setModel(new JSONModel(obj));
}
```

For displaying the *ComboBox* with provided data, it should be defined in the XML view file as in the following piece of code:

```
<ComboBox
   selectedKey="{/fSelected}"
   items="{ path: '/fItems' }">
   <core:Item key="{id}" text="{text}"/>
</ComboBox>
```

And the produced *ComboBox* element can be seen in Figure 5.

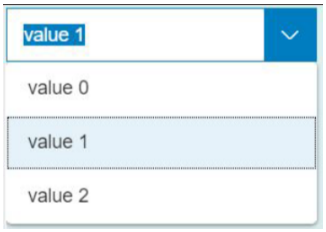

Figure 5. The produced *ComboBox*

Once an option is selected, it should be send back to the server side in JSON format, as shown in the following JavaScript code:

```
var json = this.getView().getModel().getJSON();
this.websocket.Send("MODEL:" + json);
```

The final step is decoding such JSON data in C++ and processing corresponding ROOT functions, as shown below:

```
if (str.find("MODEL:") == 0) {
   std::string json = str.substr(6);
   auto model = TBufferJSON::FromJSON<RFitPanelModel>(json);
   if (!model) {
      // failure in JSON decoding, handle
   } else if (model->fSelected == "id0") {
      // do first branch
   } else if (model->fSelected == "id1") {
      // do second branch
   } else if (model->fSelected == "id2") {
      // do third branch
   }
}
```

This simple example shows how the *ComboBox* control of *OpenUI5* can be used for item selection from a provided list. The correspondent structure for the *Fit Panel* is much more complex, but follows exactly the same logic. The main advantage of such approach - exactly the same data structures are used for communication in both directions.

5. Conclusion

As the current graphics standards are slowly vanishing, the ROOT framework should base its graphics and GUI on the technologies of the next decades. ROOT7 proposes this new modern approach by being fully web oriented and based on *de facto* standards like SVG or WebGL. Being based on a client/server model, it also allows to completely discouple the data from its graphics representation. In the same way the GUI tools are also acting on the data, and they are not embedded in the main application. The batch output remains an issue but the headless mode of the modern browsers provides an elegant solution.

6. References

[1] ROOT CERN `https://root.cern/`
[2] Antcheva I Brun R *et al.* 2011 ROOT: A C++ framework for petabyte data storage, statistical analysis and visualization
[3] ROOT6 `https://github.com/root-project/root/tree/v6-18-00-patches`
[4] About X11 for Mac `https://support.apple.com/en-us/HT201341`
[5] ROOT7 `https://github.com/root-project/root/tree/master`
[6] THttpServer Class `https://root.cern/doc/master/classTHttpServer.html`

[7] TBufferJSON Class https://root.cern/doc/master/classTBufferJSON.html
[8] JavaScript ROOT https://root.cern/js/
[9] OpenUI5 https://openui5.org/
[10] Deacon J 1995 Model-View-Controller (MVC) Architecture http://www.rareparts.com/pdf/MVC.pdf
[11] ROOT Jupyter Notebook https://root.cern.ch/notebooks/HowTos/HowTo_ROOT-Notebooks.html
[12] RWebWindow Class https://github.com/root-project/root/tree/master/gui/webdisplay/inc/ROOT