JavaScript GUI

JSROOT reloaded

Basic ideas

- The GUI (widgets) is on the client side (HTML, JavaScript, …).
- The GUI action triggers either a JavaScript function call (on the client side), or pass a formatted C++ method to be called on the server side.
- On the server side (C++) the ROOT THttpServer class is expected to handle the events.
- Users' GUI could be created using any web GUI toolkit

JavaScript GUI libraries

- Benefits:
  - Large user communities
- Strategy:
  - Select one and use it internally
  - Provide basic support for any GUI library
  - Dedicated add-ons for JSROOT/THttpServer
  - Avoid strong bounding with single one
  - Give complete freedom to the users to use their favourite one

Complex GUI

We need to build complex GUI, like editors and Fit Panel, from the C++ objects (classes) information and define the method and arguments to be processed by the server side from its class members or Getters and Setters. This is very similar to what is currently done in C++.

Graphical Editors

Objects like histograms, graphs, axis etc.. displayed in the canvas need local (on the JavaScript side) editors to change their attributes on the client side. Then a validation of the changes will trigger the update to the server side

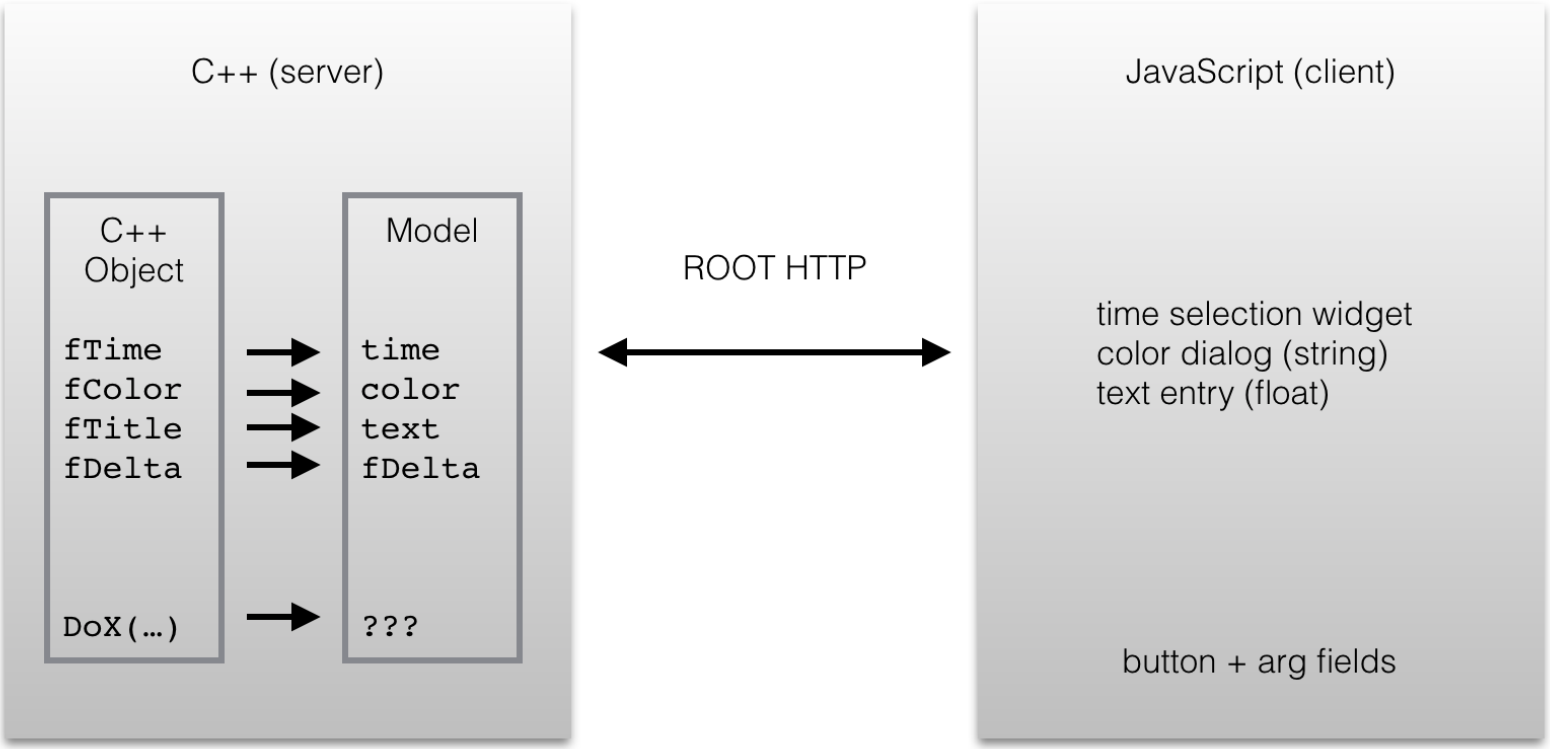
Possible options for the placement of the editors:

- On the left space (as in the browser)
- Floating, like this example
- Overlay (something like this)

User’s Defined GUI

- To implement their own GUI, users should:
  - Define the interface (model) from their C++ object to be implemented by the JavaScript GUI
  - Register the object in the http server, to make it visible/accessible on the client side
  - Create their GUI using the object's' methods/types

Interface (model) Definition



Current Status

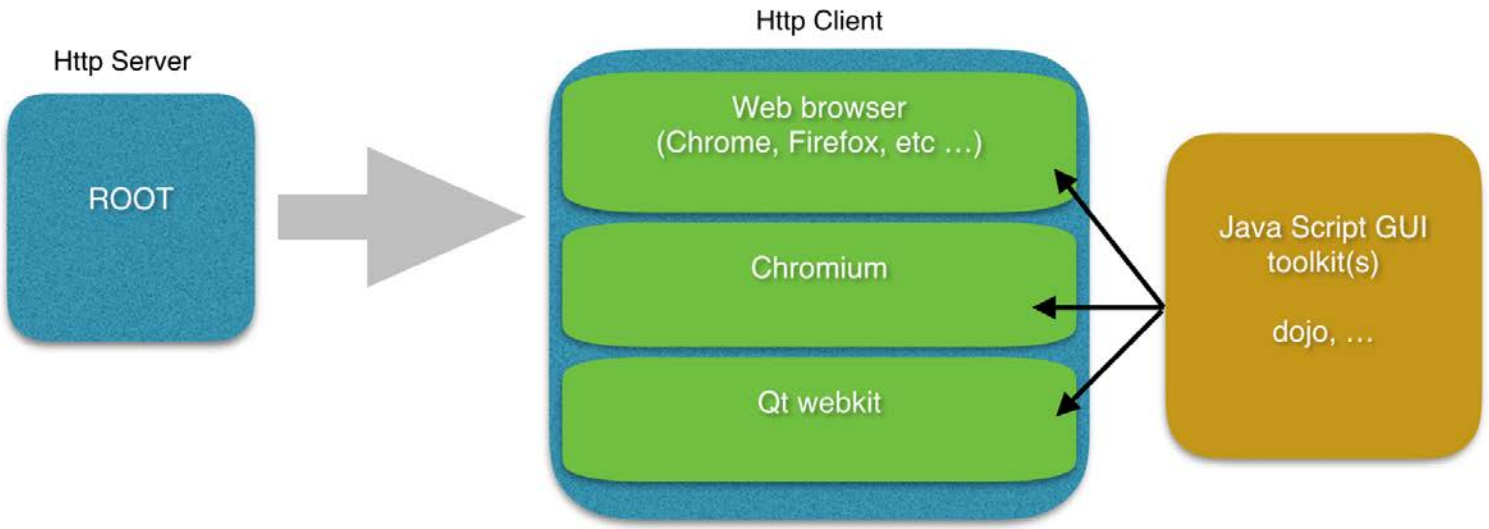
- The TRootSniffer class gives full description of the ROOT objects hierarchy used with THttpServer
- It is already possible to add objects and actions to THttpServer:

```
// register histograms
serv->Register("/", hpx);

// register simple start/stop commands
serv->RegisterCommand("/Start", "bFillHist=kTRUE;",
                      "button;rootsys/icons/ed_execute.png");

// register commands, invoking object methods
serv->RegisterCommand("/ResetHPX","/hpx/->Reset()",
                      "button;rootsys/icons/ed_delete.png");
```

Client-Server architecture



HTTP Client

- Any Web browser like Firefox, Chrome, etc... Allowing to render and interact with the graphics locally or remotely
- Local client allowing to render and interact with the graphics locally only. Examples of such tool are libchrome (from the Chromium project) and the Qt Webkit
- libchrome need investigations
- Some preliminary tests have been done with WebKit

WebKit - Preliminary Tests

- Qt4 and Qt5 implements QWebView widget, based on WebKit.
- Such widget can be integrated into any qt-based GUI as any other normal QWidget.
- Some preliminary tests with the "fancybrowser" example from both qt4 and qt5 show that JSROOT graphics in general works! One could display SVG and WebGL graphics smoothly.

WebKit Issues

- Mouse wheel events is not supported or wrongly implemented (both qt4 and qt5) - not very important
- iframe is not working (not used for JSROOT graphics)
- The web sockets works only with newest Qt 5.7.1
- MathJax.js works only with newest Qt 5.7.1
- The test was done with the existing TWebCanvas prototype. It works inside the ‘fancybrowser‘ without any modifications!

WebKit - Conclusions

There was no visible critical problem to use JSROOT graphics inside Qt. Critical functionalities are working (SVG, WebGL, MathJax.js, web sockets). Any QT application can in future include a ROOT canvas. Moreover, the class QNetworkAccessManager, could allow to redirect http requests to the local socket/pipe connection. We could provide solution for Qt users to connect widget with ROOT Canvas without creating http sockets at all. But this required more investigations and can be postponed for the future.

JavaScript GUI libraries

- There are many of them
  - Dijit (based on dojo) https://dojotoolkit.org – and soon Dojo 2 http://dojo.io
  - ext.js https://sencha.com (GPL v3)
  - Webix http://webix.com (GPL v3)
  - OpenUI5 http://openui5.org (Apache-2.0)
  - ...
- See also
  - http://stackoverflow.com/questions/200284
  - https://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks

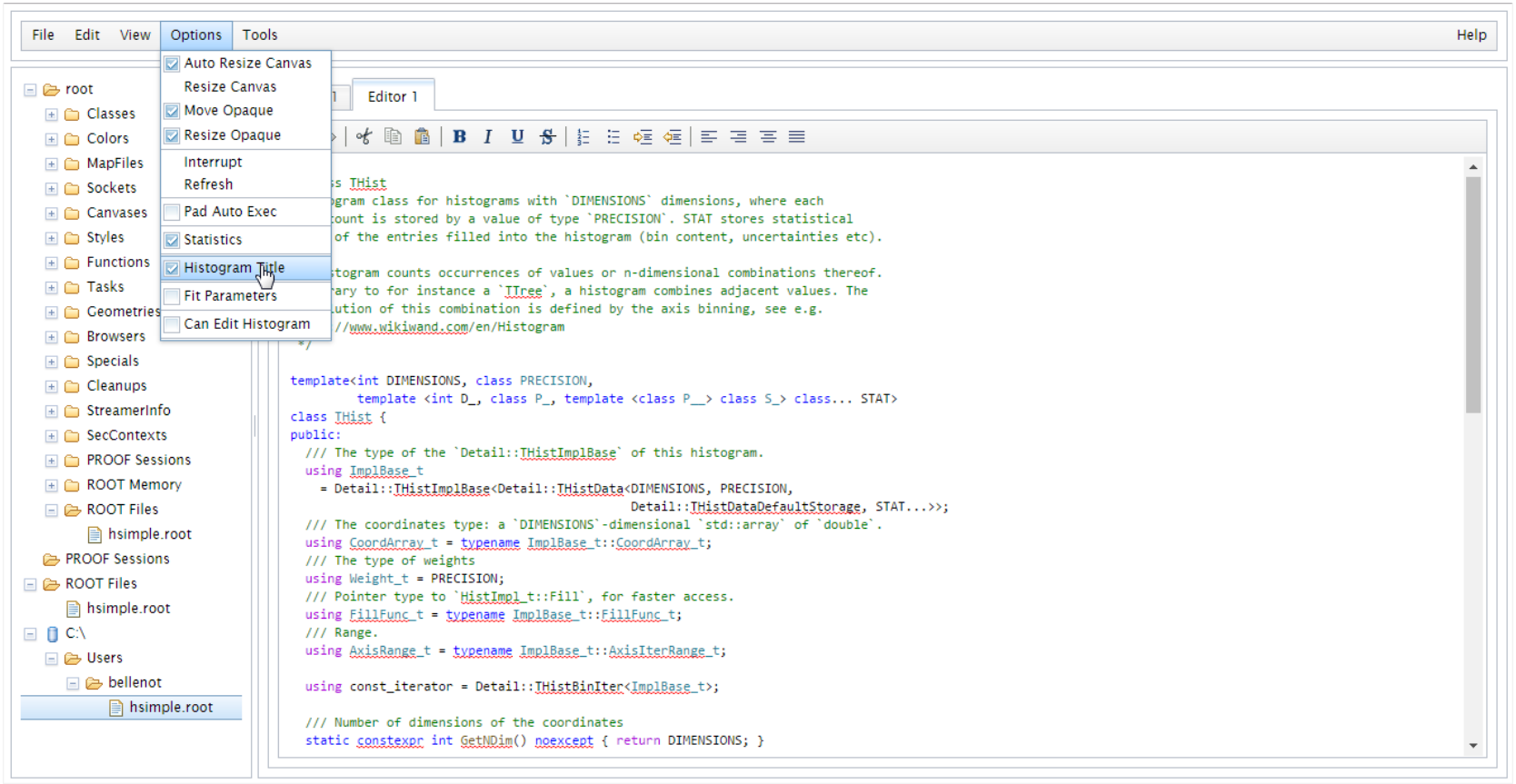
Dijit

Dijit is Dojo's UI Library (https://dojotoolkit.org/reference-guide/1.10/dijit/)

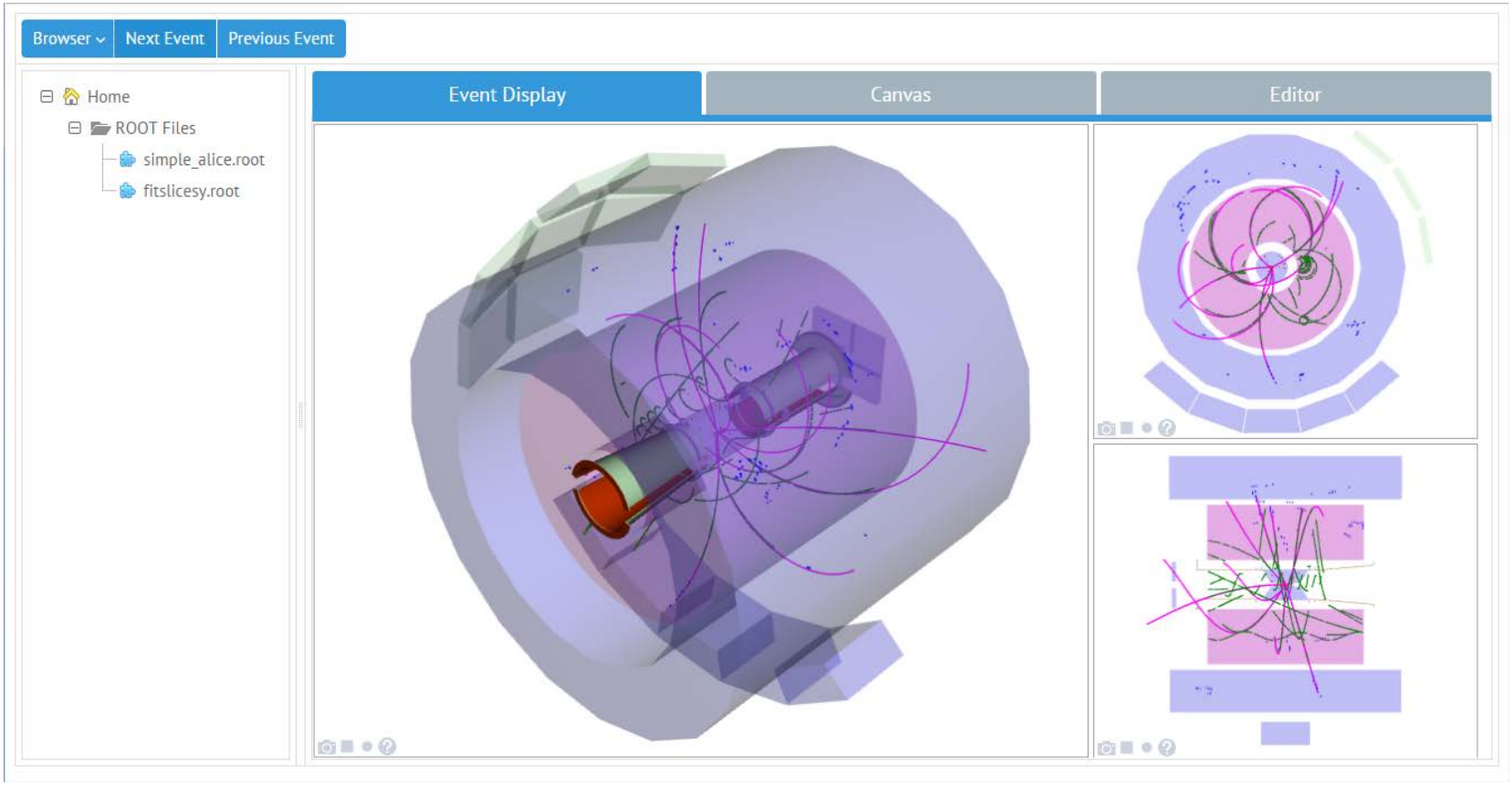
It seems the only large open-source project (beside jquery and jquery-ui), present on github and covering our needs. It offers the possibility to work in pure html, or in JavaScript

There is currently an issue with RequireJS preventing to use it with JSROOT (under investigation)

Dijit Example



Webix Example



What Next

- Select a JavaScript GUI library
- Implement wrappers (C++ and JavaScript)
- Implement a JavaScript Tree Viewer

Reserve

- On the C++ side:

```
   // register simple processline command
   serv->RegisterCommand("/ProcessLine", "%arg1% >& output_log.txt");
```

- On the JavaScript side:

```
fileMenu.addChild(new MenuItem({
   id: "exec_command",
   label: "Execute Command",
   onClick: function(id) {
     var url = '/ProcessLine/cmd.json?arg1="'+command+'"';
     JSROOT.NewHttpRequest(url, 'object', function(res) {
       if (res) {
         updateCanvas();
       }
     }).send();
   }
 }));
```

Reserve

- Another example with Webix:

```
var text = webix.ui({
   view:"text",
   id:"command",
   value:'',
   label:"Email"
});

text.attachEvent("onKeyPress", function(code, e) {
  if (code == 13) { // return
    var command = $$('command').getValue();
    var url = '/ProcessLine/cmd.json?arg1="' + command + '"';
    JSROOT.NewHttpRequest(url, 'object', function(res) {
      if (res !== null) updateCanvas();
    }).send();
    return false;
  }
});
```