ROOT7 Fit Panel – Browser

Iliana Betsou
National Technical University of Athens

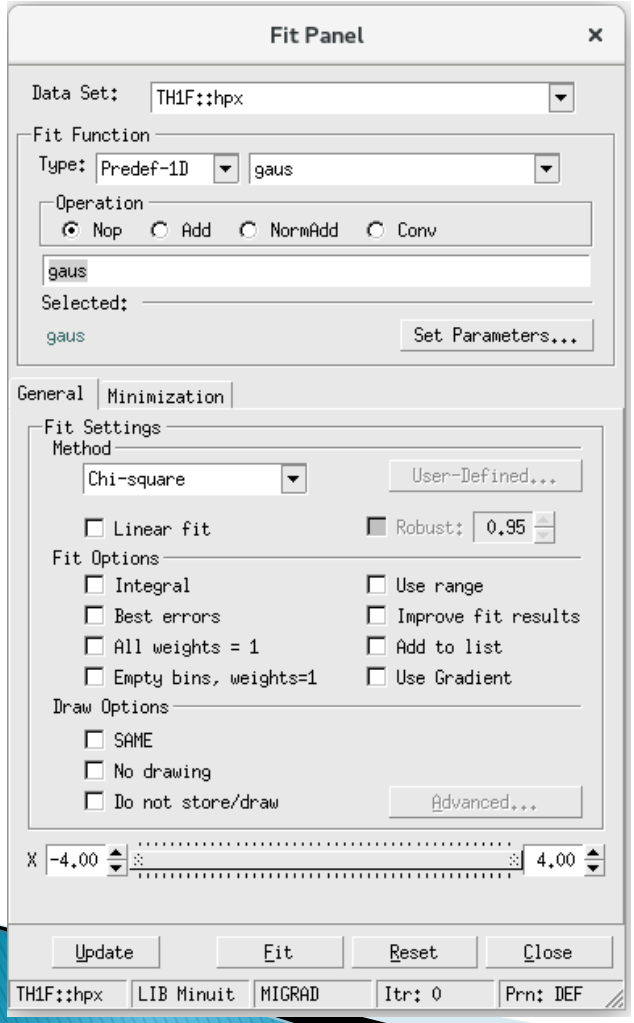
Objectives of the new version

What we want from the new version to be:

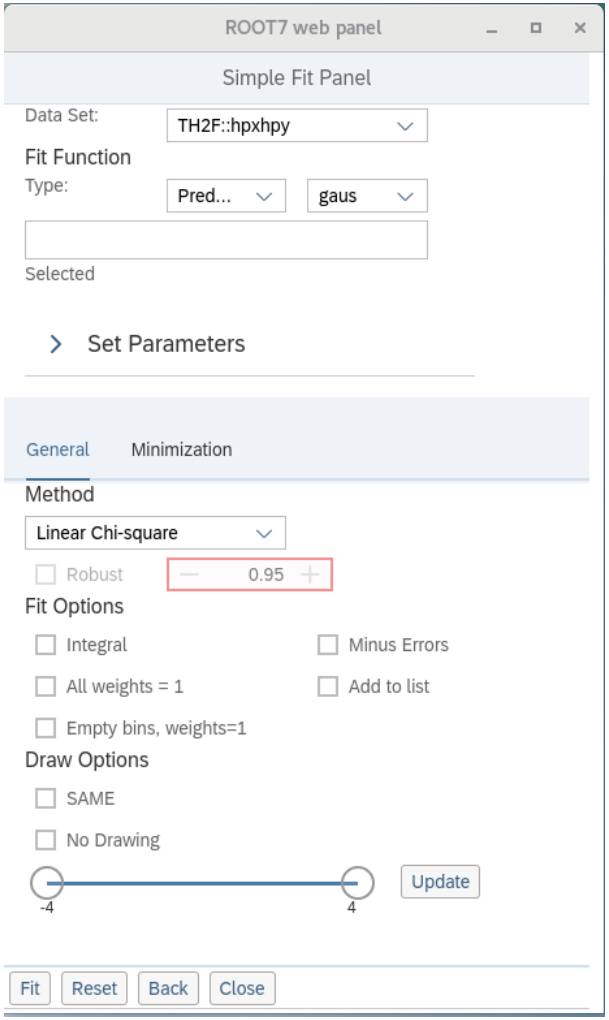
- Client/Server program
  - Web based application
  - Run directly in a browser
- Modern and user friendly
- Responsive layout

ROOT7 version – Fit Panel

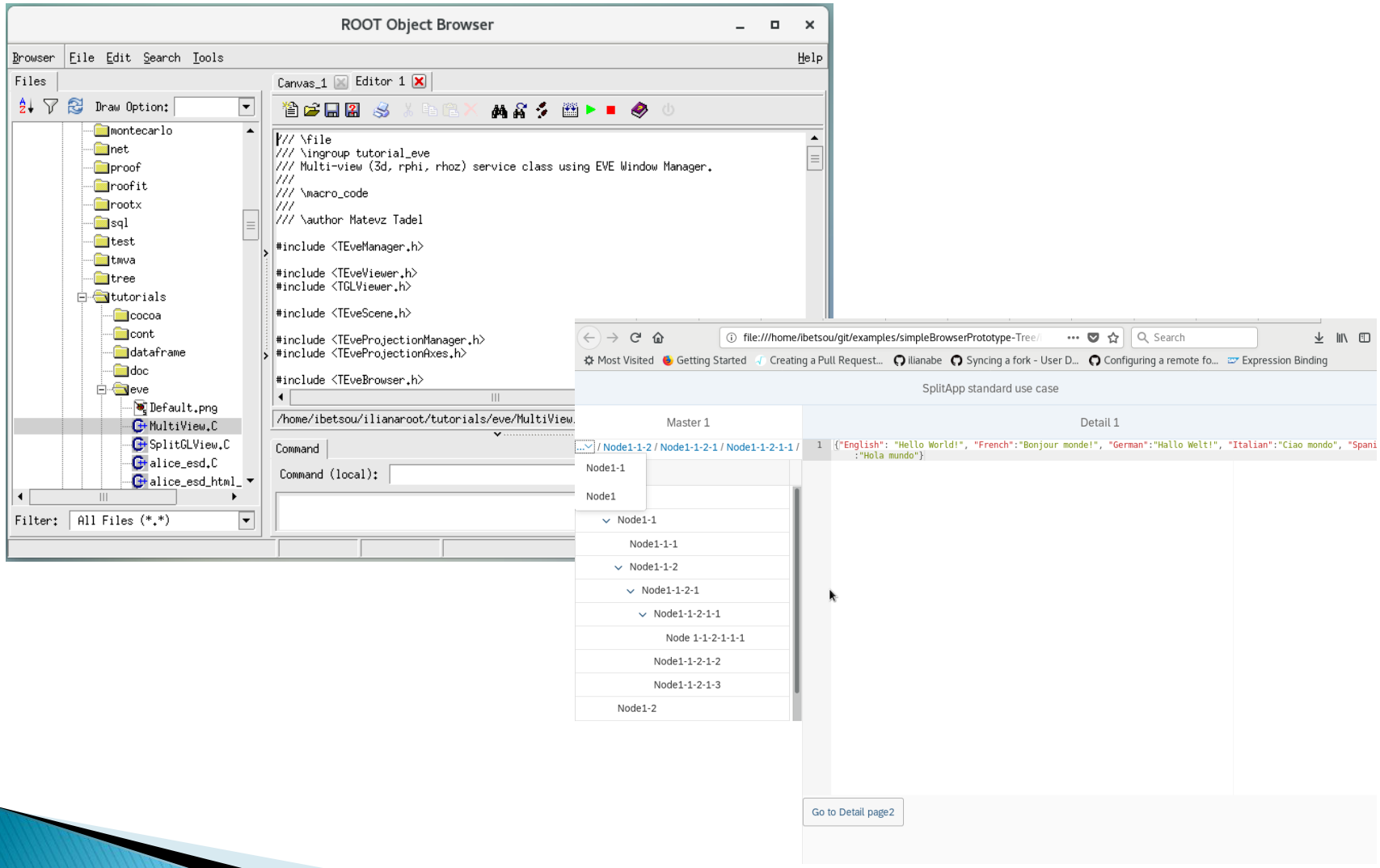
Before



After



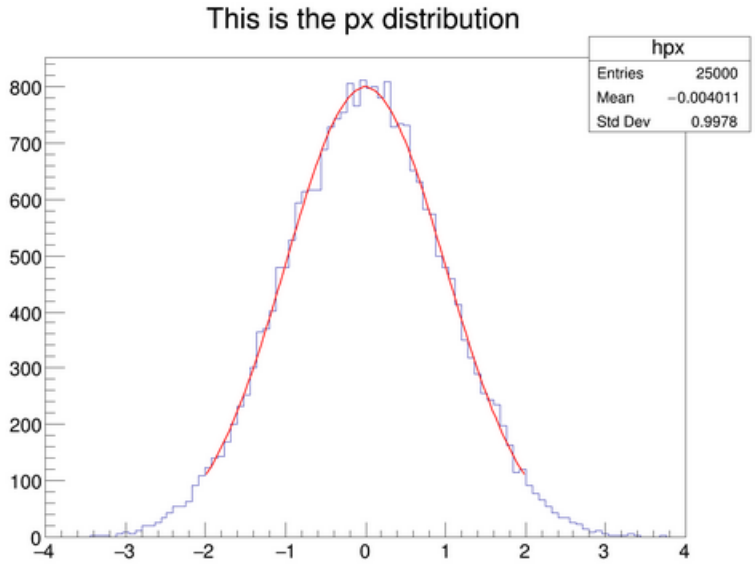
ROOT7 version – Browser



Fit Panel…

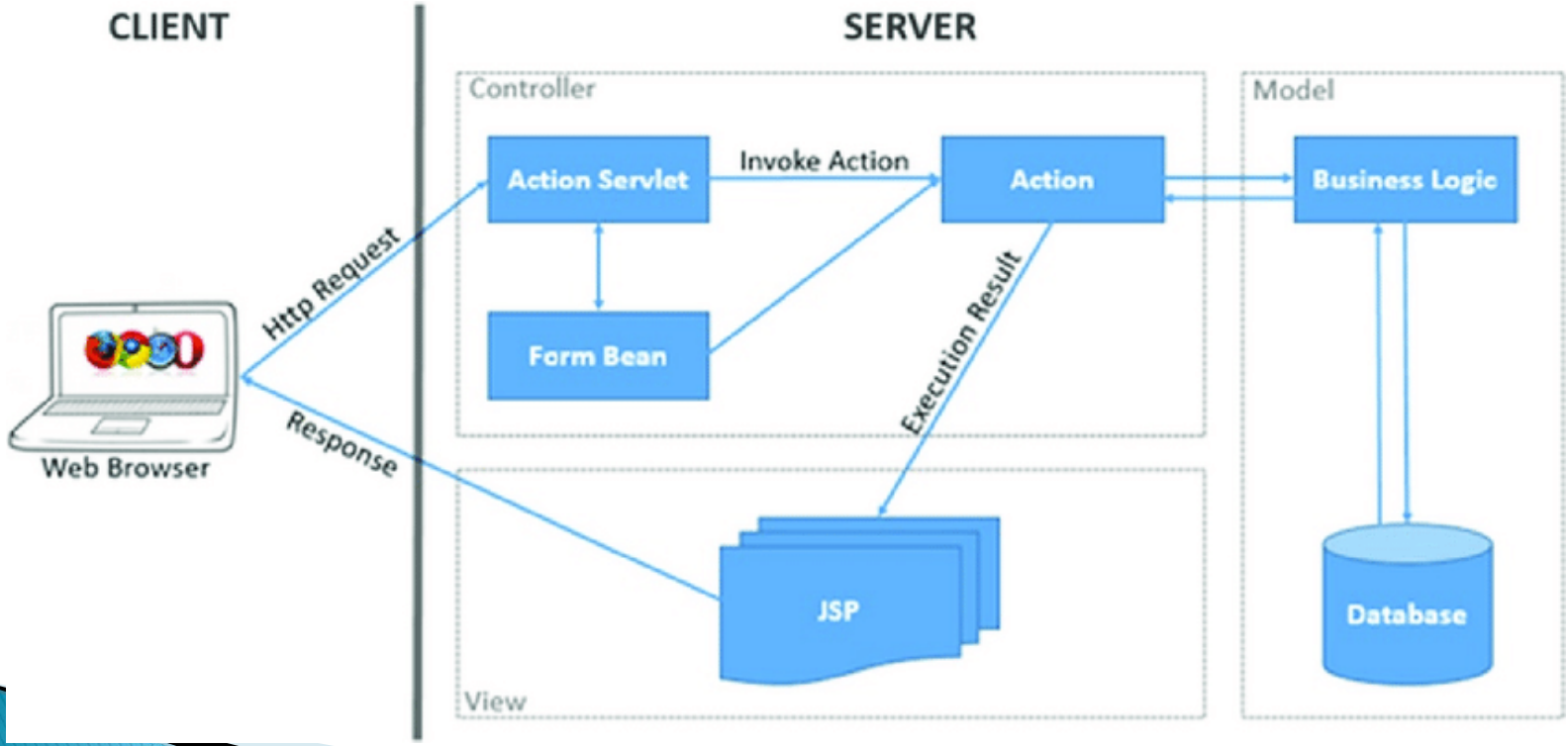
…is a GUI, which can use all the power of ROOT Fitting tools and interactively fit histograms.

- Predefined or User Function
- Draw Options
- Libraries
- Different Minimization Methods
- Range
- Print Options



Client Server Model

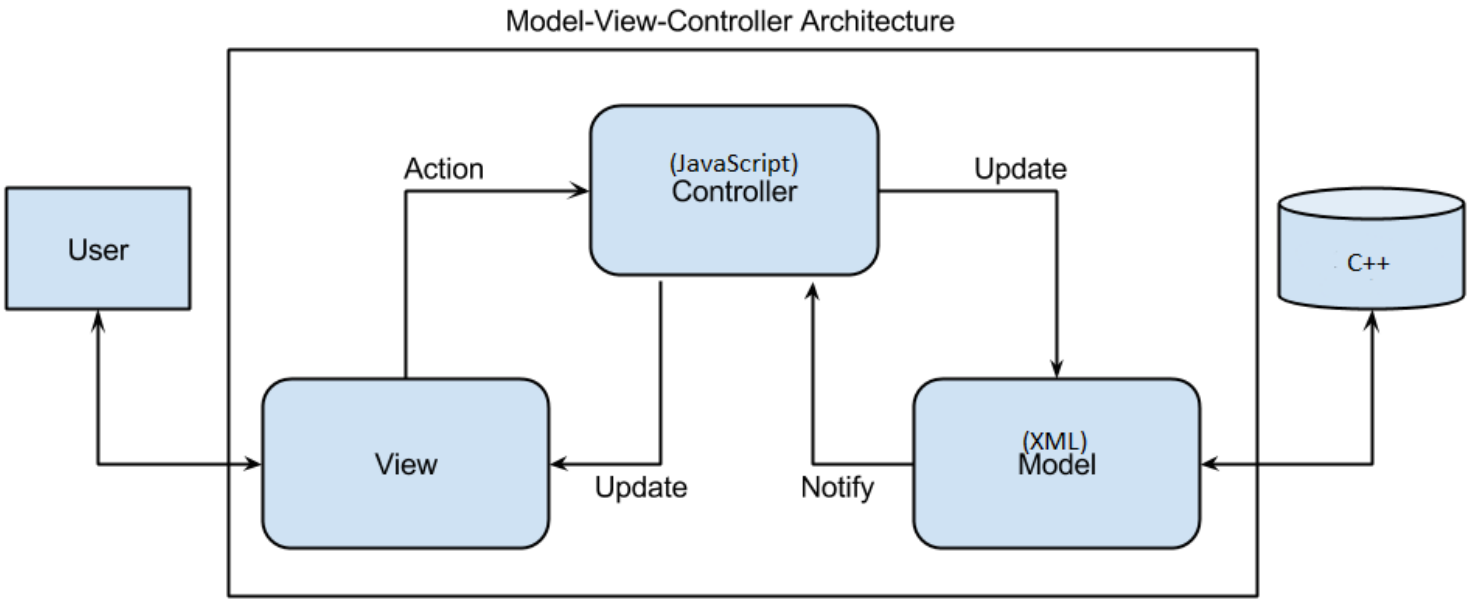
- Organizes network traffic
- Client sends requests to server
- Server responds by returning results



OpenUI5

OpenUI5 is the new technology we have used to design the new version of the Fit Panel. It is a JavaScript UI library consisting of a really large number of UI controls.

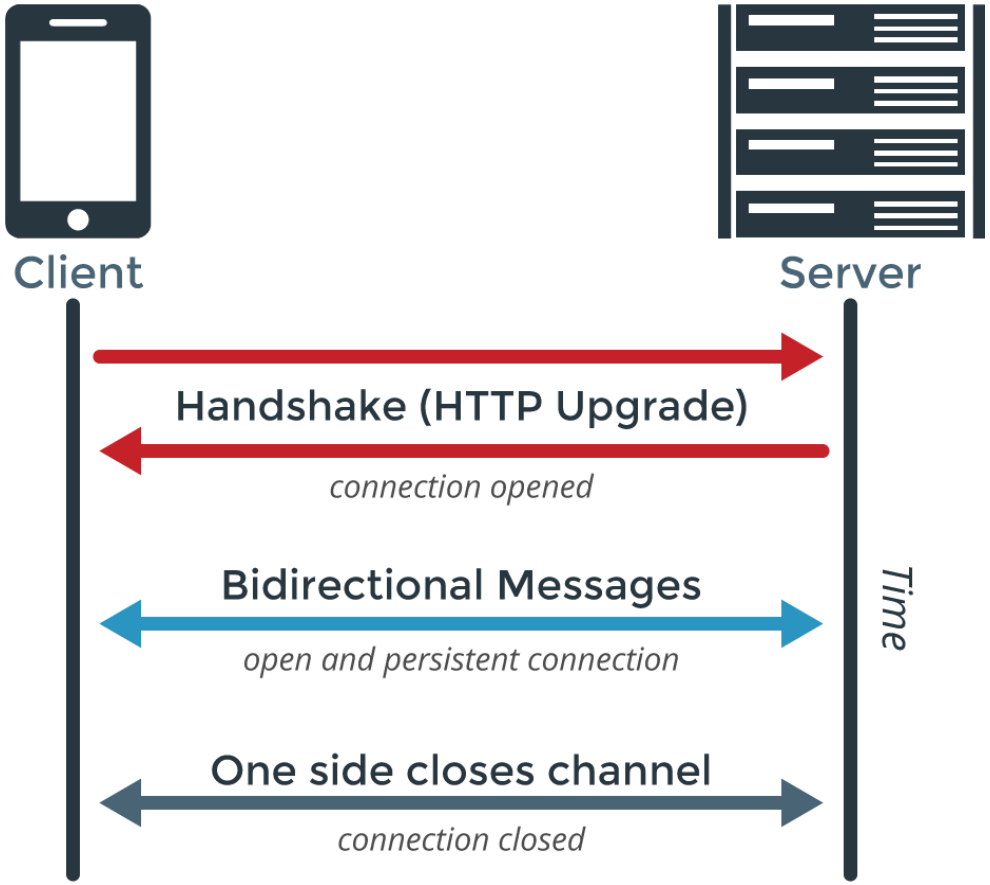
MVC Architecture



- Model Component → Dynamic data structure
- View Component → Display of the data (UI)
- Controller → Contains control logic

WebSocket Protocol

- Communication Protocol
- Duplex communication channels
- Compatible with HTTP and almost all browsers



How it's working

- Model Component – XML
  with the static values
- Server side – C++
  the non-static values
- Client side – JavaScript

- View Component
  HTML file

The Fit Panel Layout (1 /4)

The containers:

- Vbox

```
<VBox
  class="sapUiSizeCompact">
```

- Toolbar

```
<Toolbar>
```

- Form

```
<form:layout>
  <form:ResponsiveGridLayout/>
</form:layout>
```

- GridData

```
<l:Grid
  class="sapUiSizeCompact"
  vSpacing="0"
  defaultSpan="L4 M6 S10">
      <l:content>
```

- LayoutData

```
<layoutData>
  <l:GridData span="L4 M6 S3"/>
</layoutData>
```

The Fit Panel Layout (2/4)

We need more compact design:

Compact Layout: for achieving a compact layout we are using:

- Forms
- GridData
- Compact Class

```
<VBox class="sapUiSizeCompact">
```

The Fit Panel Layout (3/4)

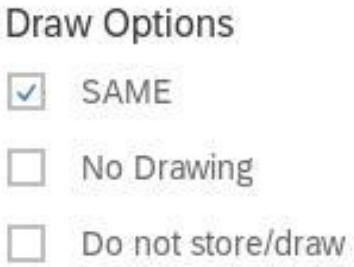
The controls:

- ComboBox

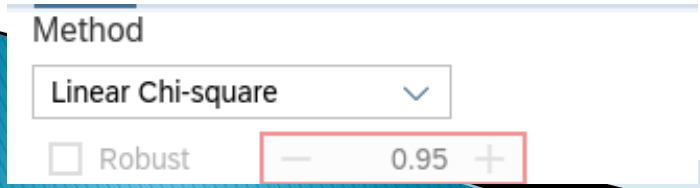
- RangeSlider

- RadioButton

- CheckBox



- Step Inputs



- Text Inputs
- Buttons

The Fit Panel Layout (4/4)

Binding: We are using data binding to communicate with the client.

```
<RadioButtonGroup selectedIndex="{/fOperation}">
   //the selected value from the RadioButtonGroup is stored
     in fOperation
```

```
<CheckBox text="Empty bins, weights=1" selected="{/fBins}"
enabled="{= ${/fWeights} !== true}"/>
  // the checkbox is enabled only if All Weights=1 option is not
    selected(false)
```

More binding

- We can copy the initial state of the FitPanel, store it in a new model, apply it on our main model and update the FitPanel.

```
resetPanel: function(oEvent){
  if(!this.copyModel) return;
  JSROOT.extend(this._data, this.copyModel);
  this.getView().getModel().updateBindings();
  return;
},
```

Server Side in C++ (1/3)

- Structures for most complicated controls (ComboBox) for implement the controls to ROOT

```
struct ComboBoxItem {
  std::string fId;
  std::string fSet;
  ComboBoxItem() = default;
  ComboBoxItem(const std::string &id, const
      std::string &set) : fId(id), fSet(set) {}
};
```

Server Side in C++ (2/3)

- FitPanelModel Structure with the definition of all types

```
struct FitPanelModel {
  std::vector<ComboBoxItem> fDataSet;

  std::string fSelectDataId;
  float fMinRange{0};
  bool fLinear{false};
  int fLibrary{0};
  bool fBestErrors {false};
};
```

Server Side in C++ (3/3)

- We give values to the parameters in our model follow the structure we have defined before.

```
FitPanelModel model;

//ComboBox for Data Set
model.fDataSet.push_back(ComboBoxItem("1", "No Selection"));
model.fDataSet.push_back(ComboBoxItem("2", "TH1F::hpx"));
model.fDataSet.push_back(ComboBoxItem("3", "TH2F::hpxhpy"));

model.fSelectDataId = "2";
model.fMinRange = -4;
model.fLinear = true;
model.fLibrary = 0;
model.fBestErrors = false;
```

Communication between C++ and JavaScript

- The communication between the server side and the client side

```
TString json = TBufferJSON::ConvertToJSON(&model,
  gROOT->GetClass("FitPanelModel"));
fWindow->Send(fConnId, std::string("MODEL:") +
  json.Data());
```

`TBufferJSON::ConvertsToJSON()` → converts an object to JSON string

Client Side on JavaScript

- Function for the communication with the server, OnWebSocketMessage

```
OnWebsocketMsg: function(handle, msg){
   if(msg.startsWith("MODEL:")){
      var json = msg.substr(6);
      var data = JSROOT.parse(json);

   if(data) {
      this.getView().setModel(new JSONModel(data));
      this._data = data; }
   }
   else { }
},
```

Fit Function on Client Side

- Button on the layout

```
doFit: function() {

   var data = this.getView().getModel().getData();
   //We get the value from the ComboBox for the function and store it func.
   var func = this.getView().byId("TypeXY").getValue();
   //We pass the value from func (JavaScript Side) to fRealFunc(C++ Side).
   data.fRealFunc = func;

   var range = this.getView().byId("Slider").getRange(); data.fRange[0] =
     range[0]; data.fRange[1] = range[1];

   //Refresh the model
   this.getView().getModel().refresh();

   if (this.websocket)
     this.websocket.Send('DOFIT:'+this.getView().getModel().getJSON());
},
```

Fitting Function on C++ (1/2)

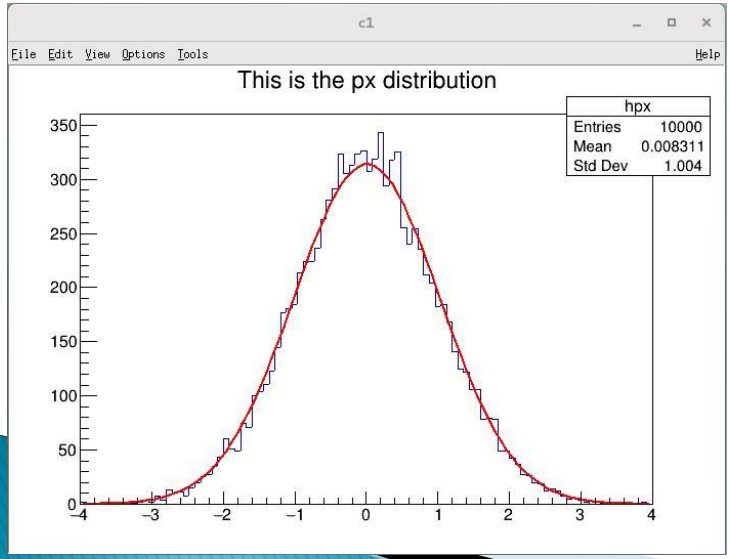
- We transform the selected values on the right form:

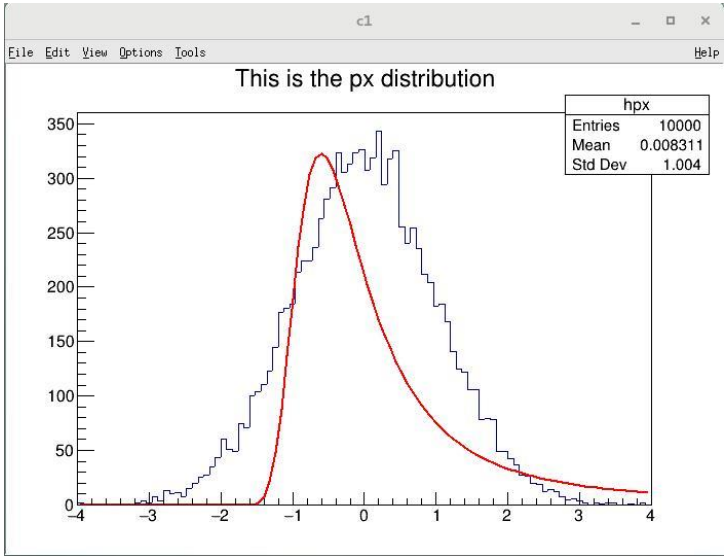
```
if (obj) {
  if (!obj->fRealFunc.empty()) {
    printf("GOT fRealFunc: %s\n", obj->fRealFunc.c_str()); }
  else { obj->fRealFunc = "gaus";
    printf("%s\n", obj->fRealFunc.c_str()); }
  if(obj->fIntegral){
    obj->fOption = "I"; }
  else if(obj->fBestErrors){
    obj->fOption = "E"; }
  else {
    obj->fOption = ""; } }
  }
```

Fitting Function on C++ (2/2)

- We assign the edited values to fit function and update

```
if (fHist) {
   fHist->Fit(obj->fRealFunc.c_str(), obj->fOption.c_str(), "*",
     obj->fRange[0], obj->fRange[1]);
   gPad->Update(); }
}
```

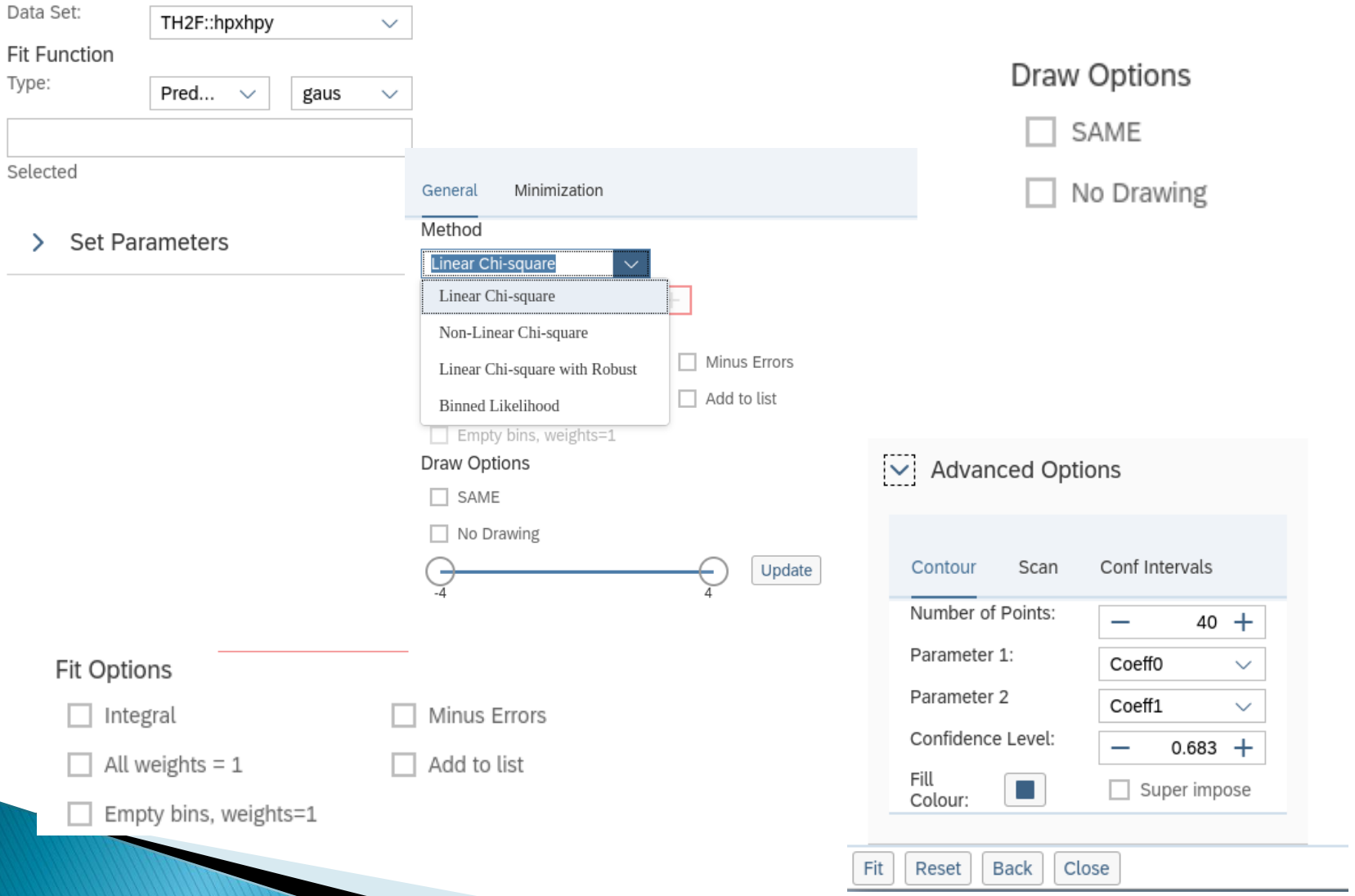




Create Window in C++

```
void Show(const std::string &where = "") {

fWindow =ROOT::Experimental::TWebWindowsManager::Instance()
  ->CreateWindow(false);

//defines the OpenUi5 element which will run on client-side
fWindow->SetPanelName("localapp.view.SimpleFitPanel");

//receive the model via WebSocket
fWindow->SetDataCallBack([this](unsigned connid, const
  std::string &arg) { ProcessData(connid, arg); });

//configure predefined geometry
fWindow->SetGeometry(450, 550);
fWindow->Show(where);

//instead showing of window just generate URL, which can be copied
  into the browser
std::string url = fWindow->GetUrl(true);
printf("Example: %s\n", url.c_str()); }
```

Changes on the FitPanel



Summary

- Model Component Creation – XML (static values)
- Server side in C++ (non static values)
- Client side in JS
- View Component

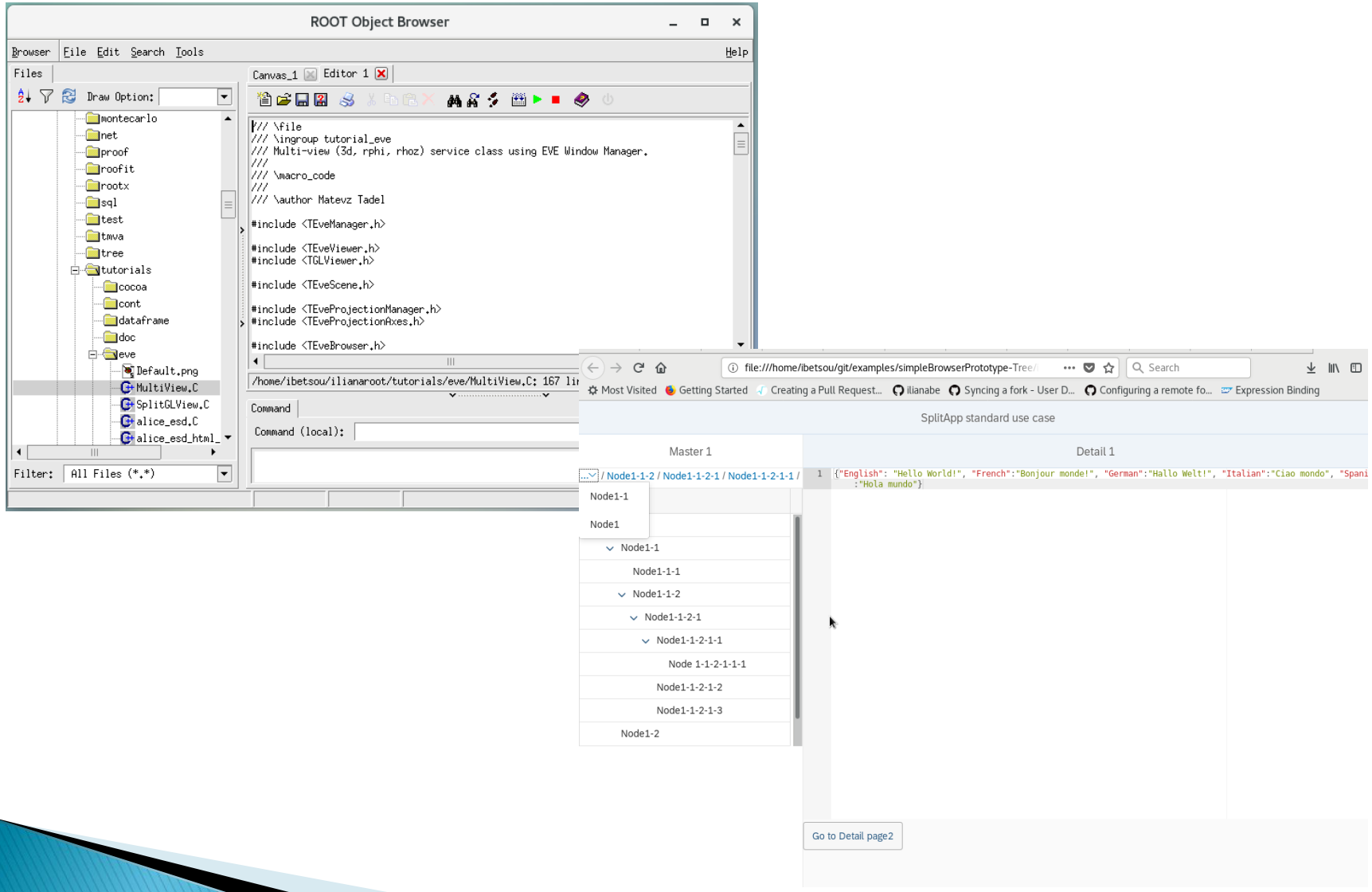
Web Based Fit Panel

Future Work

- There are a few things that still missing from the new version of Fit Panel and will be completed in the next months.

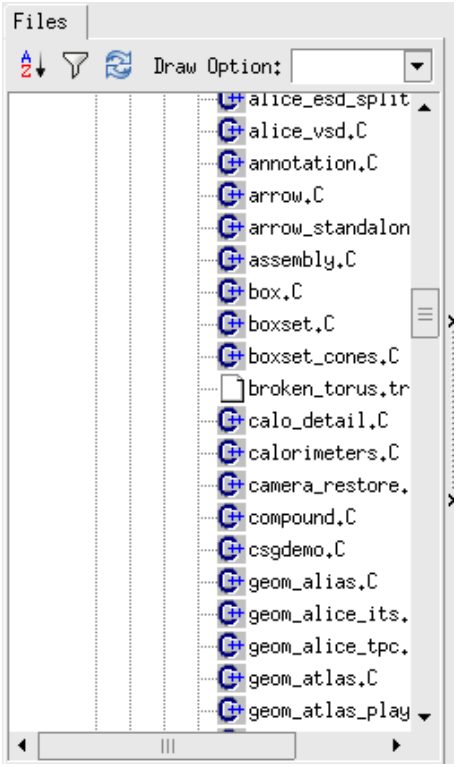
  - Picking on Rpad
  - Provide list of primitives on RPad to fit panel on C++ side
  - Add functionality to Update Range Button
  - Create a Back function

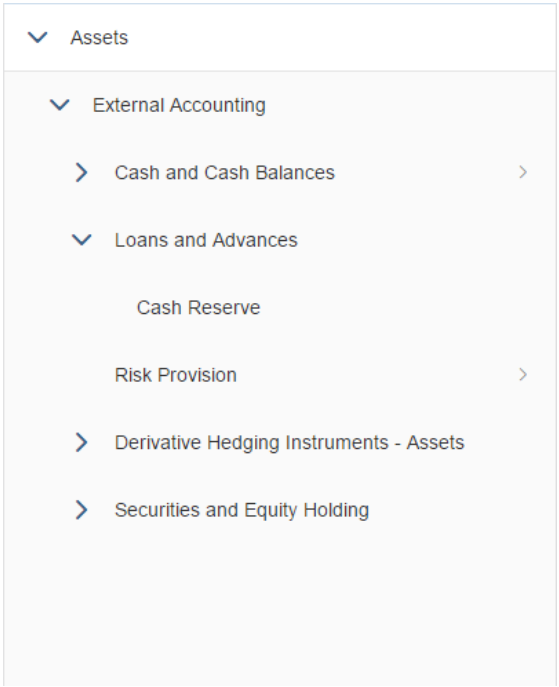
Browser Layout



The perfect control

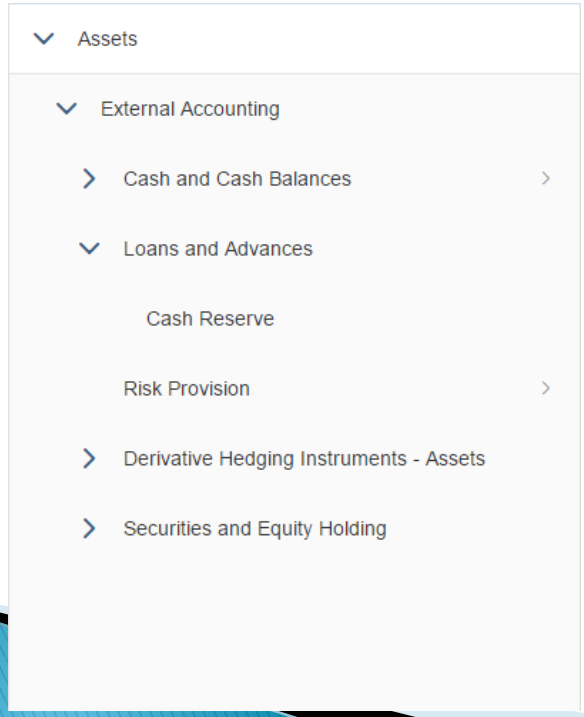
- Huge amount of data
- Easy to use
- Responsive

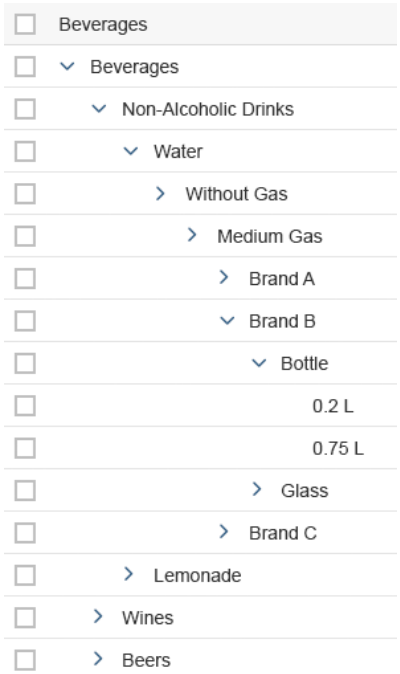




Tree vs TreeTable

- Tree is responsive but it is able to handle a limited amount of data
- TreeTable can handle any amount of data using OData protocol, but it is not responsive

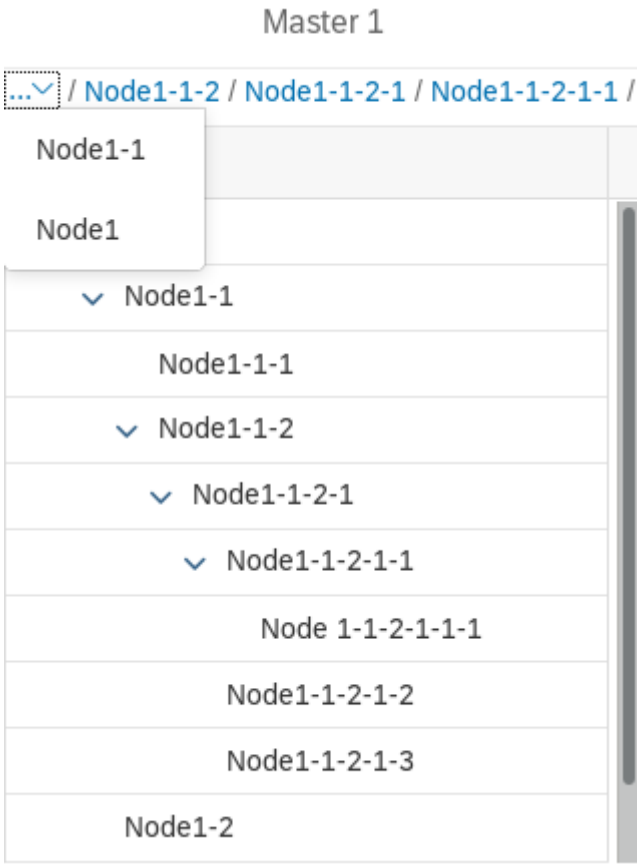




Breadcrumbs Navigation

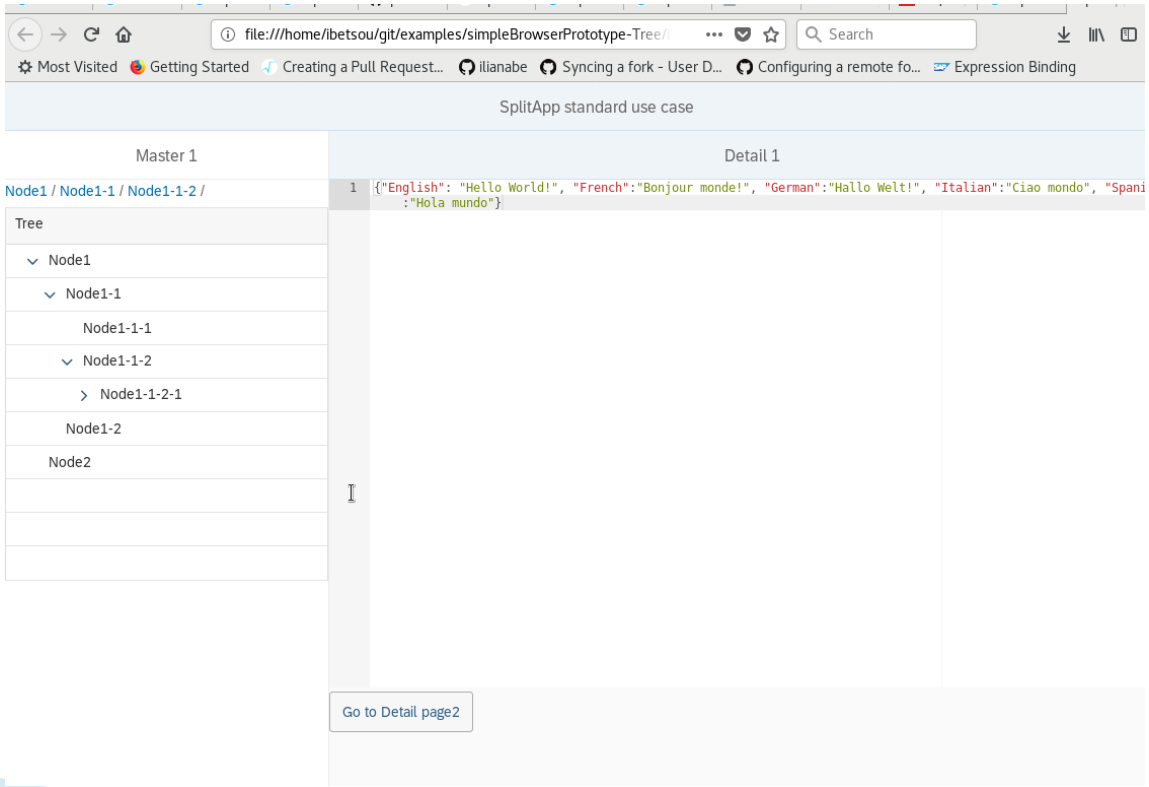
- Easier way to navigate through TreeTable

```
onToggle: function(oEvent) {
...
if(!(paths.includes(lItem))){
arr.push(sNode); rows.push(sNode, rowIndex);
}
else{
var index = arr.indexOf(sNode);
paths.length = index+1;
arr.length = index+1;
}
paths.push(lItem);
this.getView().getModel("aModel").setData(arr);
},
```

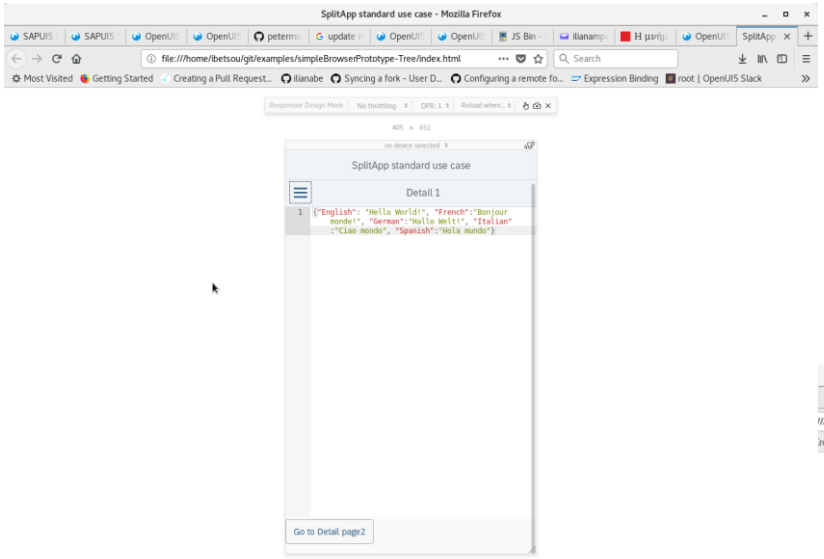


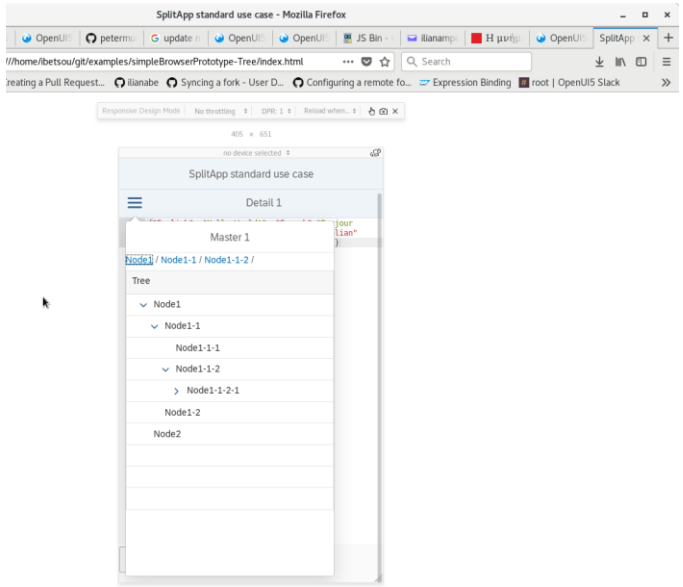
SplitApp – Responsive layout

```
<m:SplitApp id="SplitAppDemo" initialDetail="detail"
   initialMaster="master" orientationChange="onOrientationChange">
   <m:detailPages>
        <m:Page id="detail" title="Detail 1" class="sapUiStdPage">
               <m:content>
                      <m:Label text="Detail page 1" />
               </m:content>
        </m:Page>
```



Browser in smartphones





Next steps

- Fully implemented Fit Panel with all the functionalities in ROOT
- Implement TreeTable in ROOT
- Create and test a Prototype of the Browser in ROOT

More Stuff!

- Dependant Combo Boxes, Layout with code,
  Controls with code,
  MVC architecture (SAP)

The Fit Panel Layout (2/4)

The controls:

- ComboBox

```
<ComboBox id="DataSet" selectedKey="{/fSelectDataId}"
items="{ path: '/fDataSet', sorter: { path: 'fSet' } }">
        <core:Item key="{fId}" text="{fSet}" />
</ComboBox>
```

- RadioButton

```
<RadioButton id="RB1-1" text="Nop"/>
```

- CheckBox

```
<CheckBox id="Integral" text="Integral"/>
```

- RangeSlider

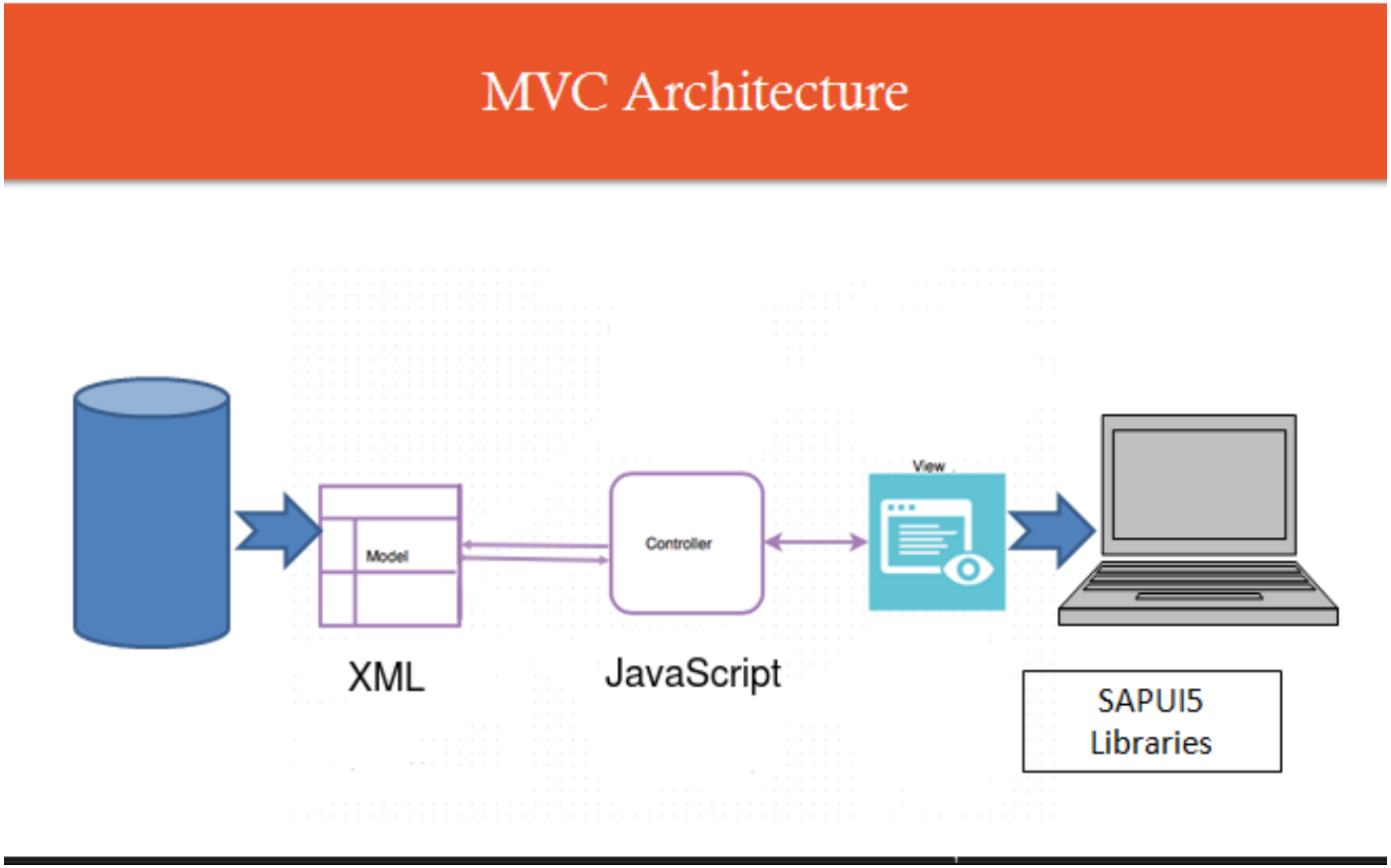
```
<RangeSlider id="Slider" range="{/fRange}"
  min="{/fMinRange}" max="{/fMaxRange}" step="{/fStep}"/>
```

Functions for handling the values

- The selected function from the ComboBox appears directly on the TextArea.

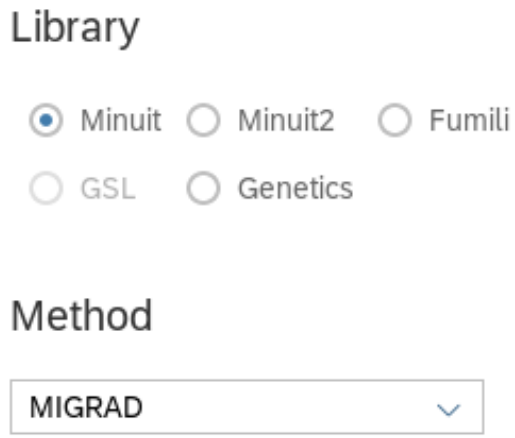
```
onTypeXYChange: function(){
  var data = this.getView().getModel().getData();
  var linear = this.getView().getModel().getData().fSelectXYId;
  data.fFuncChange = linear;
  this.getView().getModel().refresh();

  //updates the text area and text in selected tab, depending on
  the choice in TypeXY ComboBox
  var function = this.getView().byId("TypeXY").getValue();
  this.byId("OperationText").setValueLiveUpdate();
  this.byId("OperationText").setValue(function);
  this.byId("selectedOpText").setText(function);
},
```

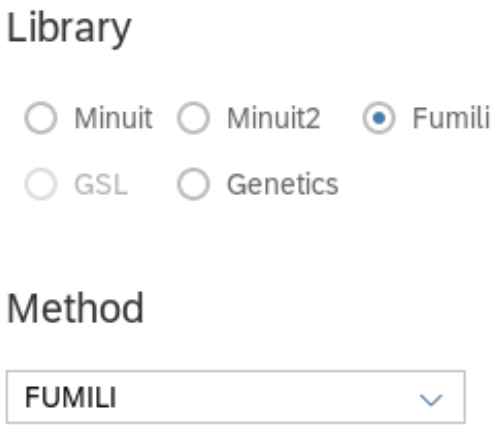
MVC Architecture



Dependent ComboBox (1/2)

There are some ComboBoxes which values are dependant from the selection of a previous control.





- Definition:

```
std::vector<std::vector<ComboBoxItem>> fMethodMinAll;
```

Dependent ComboBox (2/2)

- We give the values:

```
// corresponds to library == 0
model.fMethodMinAll.emplace_back();
std::vector<ComboBoxItem> &vect0 = model.fMethodMinAll.back();

vect0.push_back(ComboBoxItem("1", "MIGRAD"));
vect0.push_back(ComboBoxItem("2", "SIMPLEX"));

// corresponds to library == 2
model.fMethodMinAll.emplace_back();
std::vector<ComboBoxItem> &vect2 = model.fMethodMinAll.back();
vect2.push_back(ComboBoxItem("1", "FUMILI"));
```