

Project report

CERN summer student programme 2013

Development of GUIs for CERN devices

Mathias Ruggaard Pedersen

August 28, 2013

Abstract

This report documents the work that was done during a summer student internship in the CERN BE-BI-SW group in the summer of 2013. The original project proposal was to produce a java GUI to simplify the setting up of a CERN beam instrument. The end result of the project is two desktop applications for monitoring two different beam instruments as well as an Android app meant as a prototype for monitoring arbitrary beam instruments on a smartphone.

1 Acknowledgements

I would like to thank my supervisor John Fullerton for guidance during the project. I would also like to thank Stéphane Bart Pedersen and Guillaume Jacques Baud for their extremely useful Java libraries and help with these. Lastly I would like to thank CERN for the opportunity to come to Switzerland and France to work on this project.

2 Development of GUIs

During the summer of 2013, I spent two months at CERN developing graphical user interfaces (GUIs) for the beam instrumentation group. These GUIs allow users to get information about the beam instruments in real-time and thus be able to see whether a given instrument is working at its full efficiency. Three such GUIs were developed during the course of my work:

- A desktop application for a gas electron multiplier (GEM) device.
- A desktop application for an internal frequency counter (IFC) device.
- An Android app for both the GEM and the IFC device, but also intended to be compatible with any arbitrary beam device.

All of these GUIs were developed in Java, and the first two were developed using the CERN Front-End Software Architecture (FESA) framework, which provides an easy way to build GUIs with a similar look and feel.

The source code for the applications is available at [2].

2.1 GEM desktop application

GEM detectors are a kind of particle detector. They consist of a gas-filled chamber with a read-out board in one end of the chamber. When a particle moves through the gas, it ionises the gas, producing electrons. Due to an electric field in the gas chamber, the electrons drift toward the read-out board, and pass through GEM foils on the way. GEM foils are thin plates with tiny holes in them. When a voltage difference is induced between the two sides of the GEM foil, a very strong electric field is generated in the holes, and when the electrons pass through these, electron multiplication occurs. Thus, when the electrons reach the read-out board, they will have multiplied to such a degree that a current is induced on the read-out board, and this can be picked up as an electrical signal.

In the device I have been working with, the read-out board is a 2D read-out board, meaning that the device outputs both a horizontal and a vertical read-out, and two different graphs are therefore needed to display the information.

The purpose of this GUI (and the others for that matter), is to display the read-out data visually as well as present other information and data from the device in a way such that a user can quickly get an overview of the status of the device and determine whether there seems to be any problems with it.

The finished application can be seen in figure 1. On the left-hand side of the application is a list of information, most of which comes directly from the device. The maximum and average values are computed by the application from the read-out data, however. All of the information is updated dynamically whenever the application receives a new update from the device.

The middle of the application contains the graphs showing the vertical and the horizontal read-out from the device. These graphs were made using *cern.fesa.expert.guicomponents.plot.plot2d.SimpleNew2DGraph* library, which automatically generates a graph from an array of data passed to it. The graphs come with a lot of built-in functionality like zooming in and out and choosing different kinds of curves.

The “Fetch data” button allows the user to retrieve data once from the device and update the GUI with this new data. The “Subscribe”/“Unsubscribe” button allows the user to continually receive new updates from the device, with the GUI automatically updating when the new data is received. The “Clear graphs” button removes all data from the graphs. Apart from these

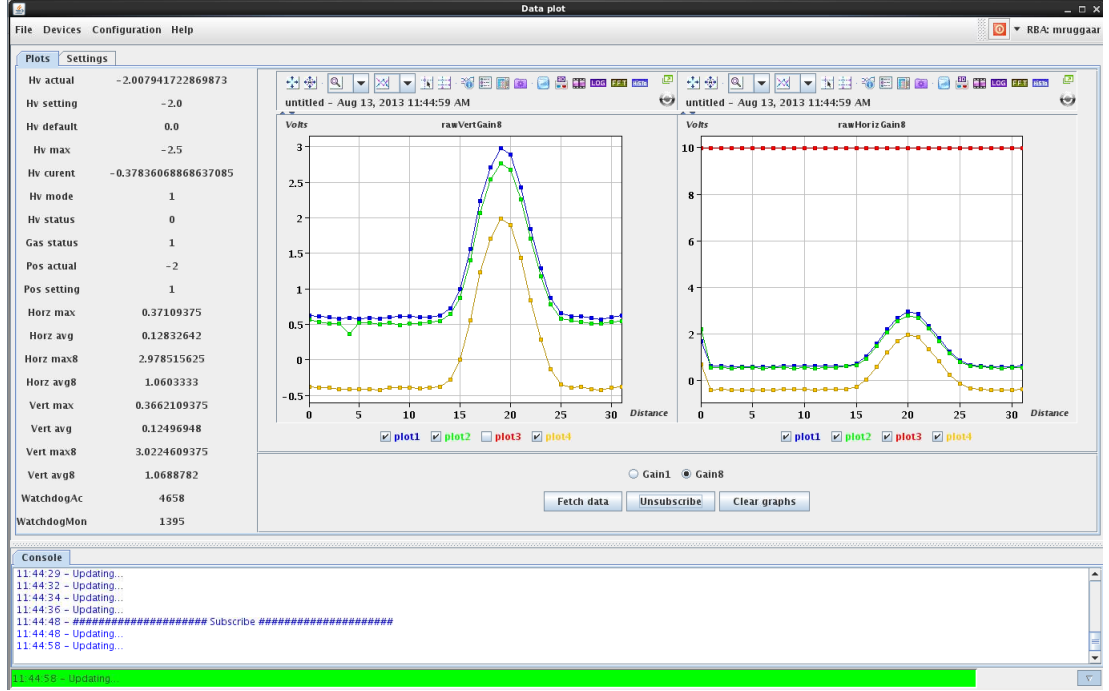


Figure 1: The main tab of the GEM desktop application.

buttons, the checkboxes determine which of the plots to be shown in a given graph, so if the user is only interested in one of the plots, they can uncheck all the other plots and only display the plot of interest. The two radio buttons allow the user to change between the amplified (gain 8) signal from the device or the unamplified signal (gain 1).

The settings tab (not shown) contains a number of different input fields, mainly for noise reduction of the read-out signal. It also contains an input field for the buffer size as well as a series of checkboxes that allow the user to select which columns of the graph to hide from the graph.

2.2 IFC desktop application

The IFC desktop application is fairly similar to the GEM desktop application, and their purposes are largely the same. A screenshot of the IFC desktop application can be seen in figure 2. On the left-hand side is again a panel showing various information about the device. Unlike the GEM desktop application, the IFC device only has one plot per graph, so it is not necessary to have the checkboxes for determining which of the plots to show.

The information in the left-hand panel is generated dynamically. Unlike the GEM desktop application which only shows a fixed number of values, the IFC application was made to be a bit more generic, so it simply shows

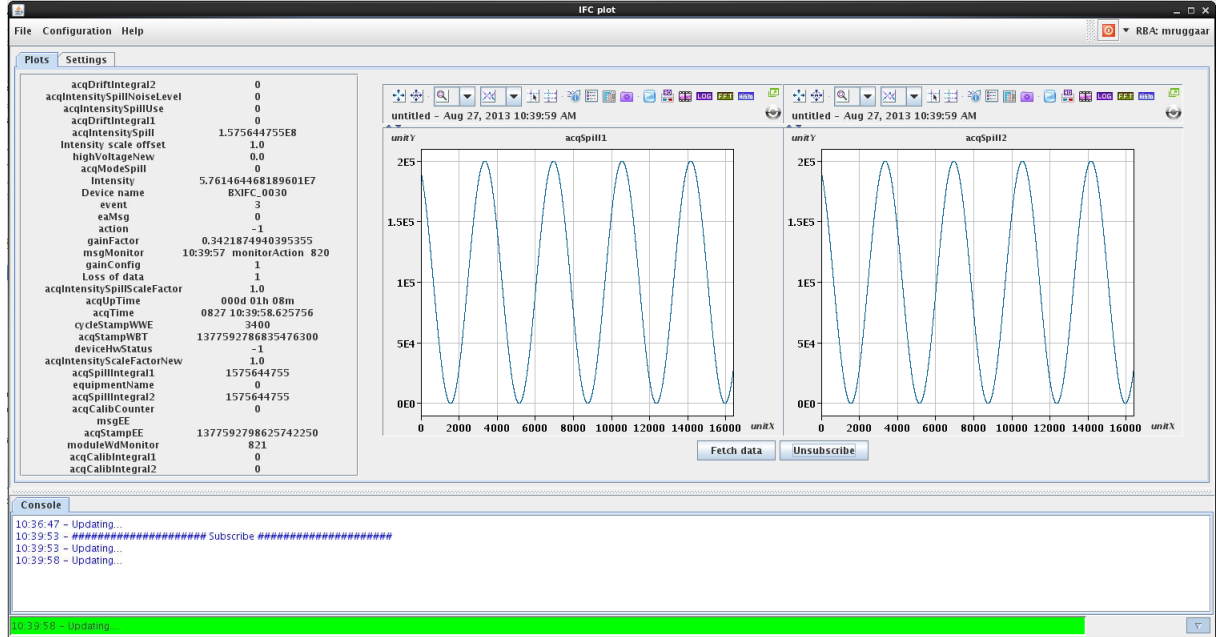


Figure 2: The main tab of the IFC desktop application.

all information which is not in the format of an array, since these would take up too much space in the panel.

Like the GEM desktop application, the IFC application also has a settings tab which is not shown here, where values for noise reduction can be set.

2.3 Android application

The main challenge in making the Android application was figuring out how to get the data from the device and onto the Android phone. In the end, it was decided to do this by having a desktop application read data from the device and then write this data to a public web server, from where the Android phone can then download it and display it. The data on the public server must be written in a specific format in order for the Android application to read it properly.

Currently, the GEM desktop application also handles this task of writing to the web server when it reads the data from the device. In this way, the GEM desktop application has two distinct functions, which is not a very good solution, and the two functionalities (displaying data and writing data) should be split out into two separate applications instead. The GEM desktop application is also the only application that has been made to write to the web server so far, so only this one device can currently be displayed on the Android app. More applications for writing device data to the web

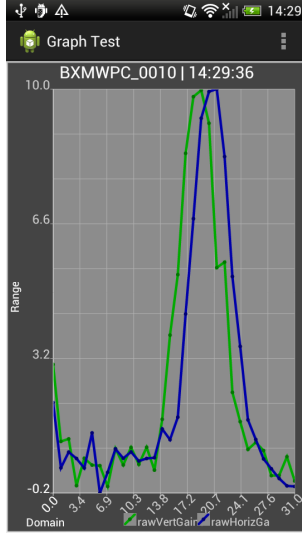


Figure 3: Graph display.

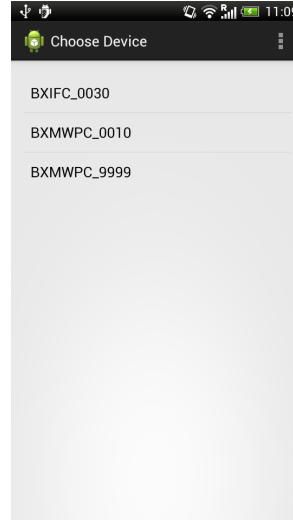


Figure 4: Choosing a device.

server would have to be written in order to add additional devices to be shown on the Android app.

When the data is on the web server, the Android app can easily read it and display it as a graph, as seen in figure 3. The graphs themselves are created using the free open source library AndroidPlot [1].

The menu button in the top right-hand corner brings the user to another screen where they can choose which device to display data from. This screen can be seen in figure 4. This screen reads the HTML page of the public web server to see all the text files that are hosted on it, and then displays the name of each text file in a clickable list. In this way, as soon as text files are created on the web server with data from a given device in the correct format, the device will automatically be added to the device list on the Android app without changing any code.

3 Future work

Both the desktop applications are pretty much done and functional in their current state, apart from a few features such as the noise reduction. However, neither of them have been tested with an actual beamline, so it's not certain that they will work during operation. Also, since no end users were around during development to test usability and give feedback, there may be some features currently missing from the GUI that the end users will require.

The Android app is meant merely as a proof of concept, and is therefore still lacking a lot of functionality. It currently has the ability to display

graphs and change device, but nothing else. An improved version of the app should also have the ability to display plots in separate graphs, especially when displaying several different plots. It should also be capable of displaying not just graphs, but also device information, like the information shown on the left-hand side of the desktop applications.

There was a bit of discussion concerning how the transferring of the device data from the device to the web server should be done, without any consensus being reached. The current system works such that each device is responsible for putting its data on the web server, which can for example be done using a small script run by the server that handles the device. This means that many small scripts have to be written in order to add many devices to the service. However, the advantage is that all the data for the app is uniform, and can therefore be handled generically and easily by the Android code. The format currently used for the text files on the web server is a home-made format that was quickly put together. If this system were to go operational, it would be a better idea to use a more standardised format, such as XML.

Some people did not like this approach, and preferred instead a system in which the user would have to know the name of the device and property that they wanted displayed. Then the user would enter the device name and property name, and the app would ask the device for this information and display it. Both of these approaches are possible, as are other approaches. Some other concerns were that it would be necessary somehow to direct access to some devices, so that a given user can see just a subset of all the available devices. A system for enabling this would have to be put in place. All in all, there are a lot of different interests and preferences that come into play when designing such a system, and if it were to go operational, these would have to be weighed against each other.

References

- [1] Androidplot.com — an android api for creating charts and plots. <http://androidplot.com/>. 5
- [2] Source code. <https://drive.google.com/folderview?id=0B-2Ti7twcspdc0pMRjFCOW0zUEk&usp=sharing>. 2