



DEVELOPMENT OF CONTROL SOFTWARE FOR PETRA III BEAMLIN

Aarón J. Alejo Alonso (Universidad de Salamanca, Spain)*

Summer 2012

Supervised by Hasan Yavaş

Abstract

The aim of this report is to document the developing process for *BLC*, the new software that should be used for the control of beamline P01 in Petra III. The most important decisions taken during the deployment will be explained. Also a brief introduction to the usage of the program and some future guidelines are shown.

*aaron.alejo.alonso@gmail.com

Contents

1. Introduction	5
1.1. Motivation	5
1.2. Experimental Setup	5
1.3. Current Software	6
2. User's Manual	10
2.1. Launching <i>BLC</i>	10
2.2. Main window	12
2.3. Real-Time Plots	14
2.3.1. The plot area	14
2.3.2. Change attribute plotted	15
2.3.3. Modify plot	16
2.3.4. Autoupdate axis / Refresh	17
2.3.5. Save Plot	18
2.3.6. Maximize Plot	18
2.4. Change an attribute	19
2.5. Scan device	20
2.6. Run script	21
2.7. Plot Viewer	23
3. Programmer's Manual	25
3.1. Introduction	25
3.2. Used Tools	25
3.3. Classes	26
3.3.1. Device Threads	26
3.3.2. Interface Classes	29

3.3.3. Model classes	29
3.3.4. Other classes	34
A. FIO File Format	37
B. Config File	39

1. Introduction

1.1. Motivation

In 2009 Petra III started user operation as the most brilliant storage-ring-based X-ray radiation source in the world. Although most beam lines accept general user proposals for experiments, some experimental stations are still being set-up, such as the inelastic x-ray scattering (IXS) end station of the Dynamics Beamline P01.

There will be two different types of IXS setups available for users. The software currently in use, online, as it will be shown briefly, is versatile and powerful, but it might get complicated with the two setups while unintentionally allowing non-experienced users to change settings such as calibrating the motor value. In order to make the control of the setup user friendly and at the same time to prevent the "unauthorized" users from accessing the advanced controls, a simple and customizable tool was needed.

1.2. Experimental Setup

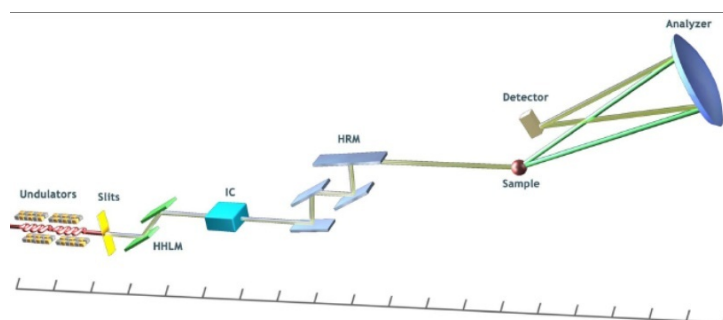


Figure 1: IXS Setup

As *BLC* is going to control the devices that are part of the experimental setup and there will eventually be more than one setup, the first thing we are showing is the experimental

setup. *BLC* will offer the user the possibility to choose which setup to be used, and showing a scheme of the different setups depending on the choice.

A typical IXS setup is shown in figure 1. The devices that *BLC* can currently control are the ones before the sample.

An image of a diced analyzer is shown in figure 2

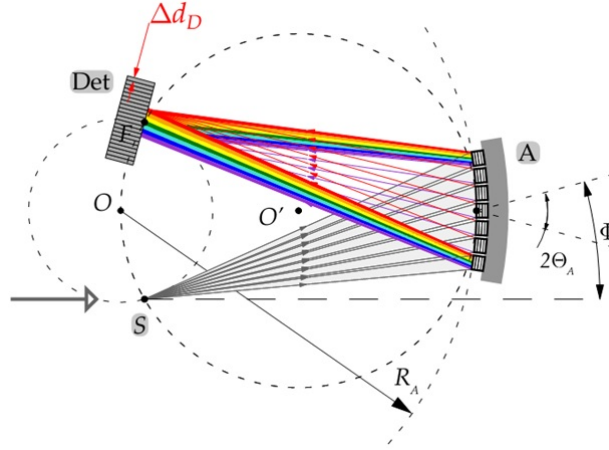


Figure 2: Diced Analyzer, *Shvydko, Y. et al. submitted*

1.3. Current Software

The beam line control is currently managed by, *online*, which has been developed and upgraded for some years, making it a powerful and versatile tool. Here, we will show some screenshots, and explain some differences between *online* and *BLC*, taking into account two of the most important window in common: main window and dialog.

In the figure 3 you can see the Main Window for *online*. As you can see, most of the information is shown there for current values. Anyways, some changes have been made to create the new tool,

1. Instead of showing the name of the motors, they appear in the screen as attributes of devices important for the experiment. There is a big disadvantage with this point

of view: only a few of them can be shown. Anyways, it may be interesting to focus on some of them, making the general window clutter free.

2. For *online*, autorefreshing the values of the attributes shown is an option. In *BLCit* is a mandatory fact that they will be updated regularly.
3. Due to the fact that online can be used via command-line, there is a box for such activity that is not offered in the new one.

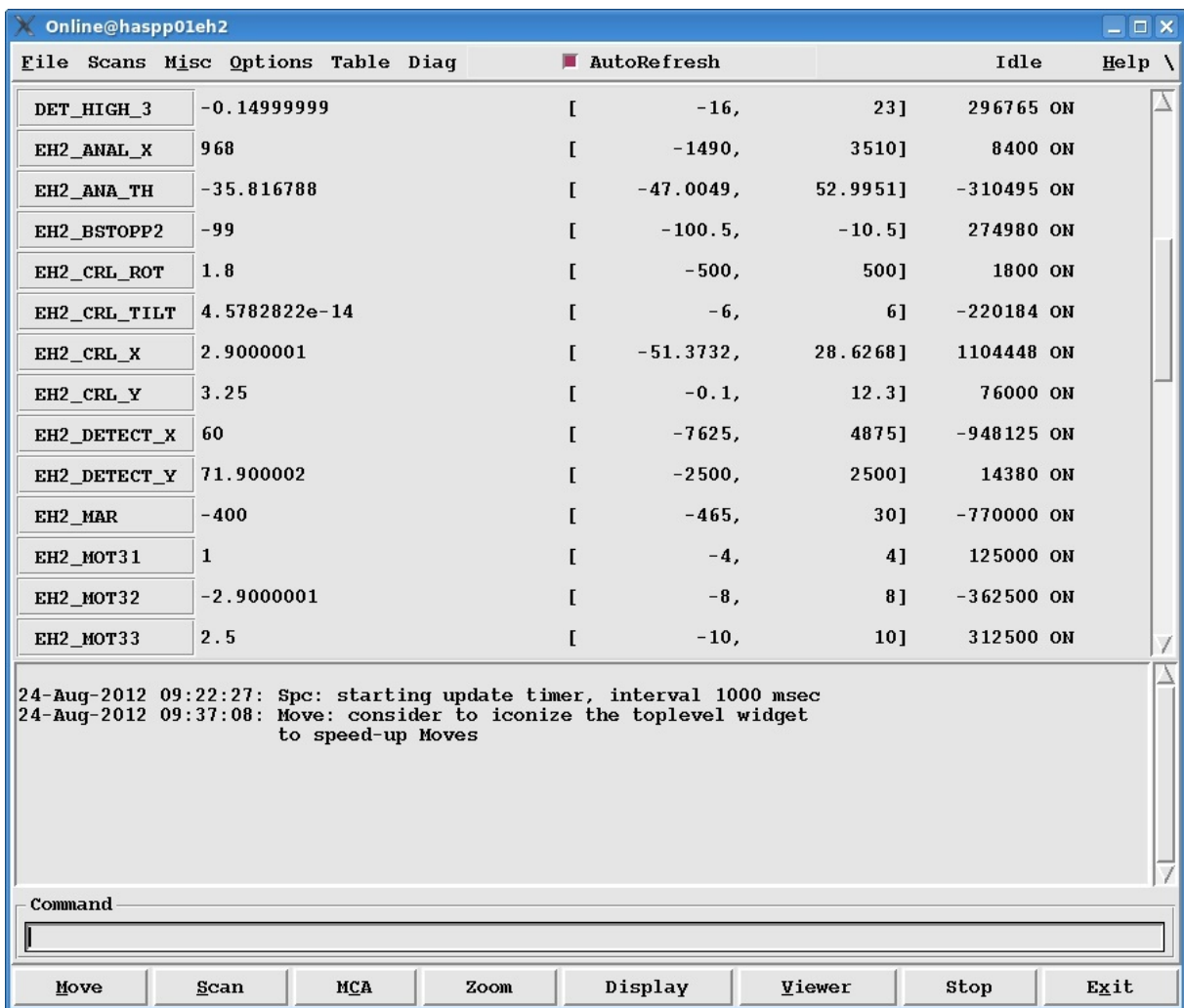


Figure 3: Online - Main Window

In the figure 4 you can see the dialog that online shows when a scan is going to take place. Let's check some of the differences

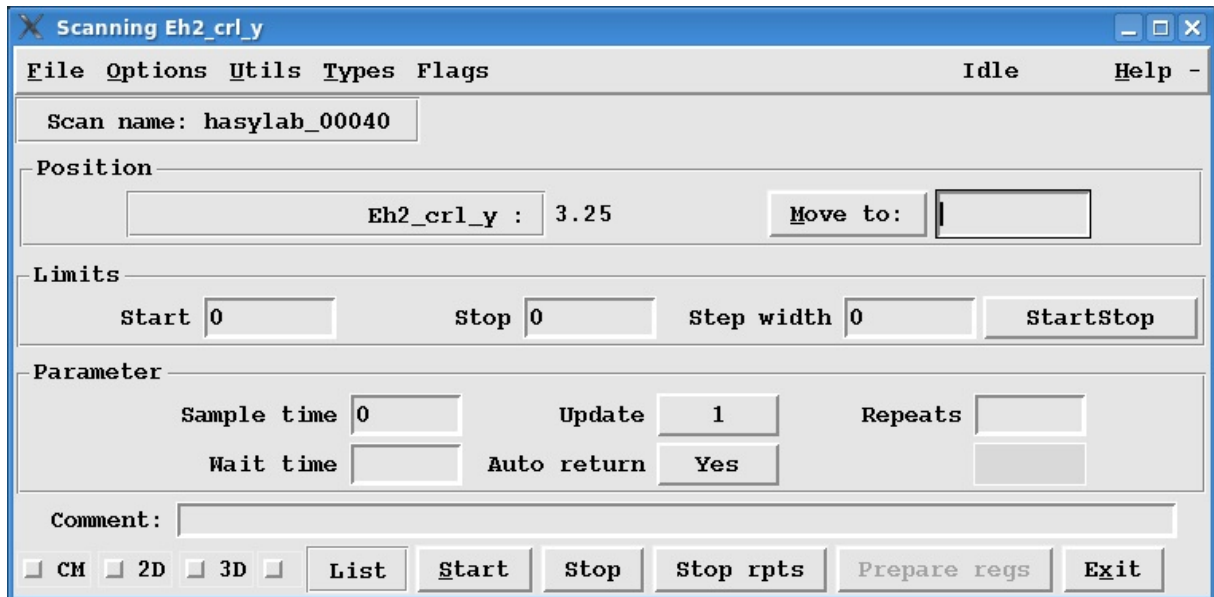


Figure 4: Online - Scan Window

1. In this case, most of the options offered by online are included in *BLC*. Still some of them are missing, but not the most important ones for the standard operation of the beam line .
2. *BLC* will plot the scan in the same window where the options are chosen, while in *online* they appear in a new window. The advantage for that option is that more than one plot can be done at the same time. Our tool solves this problem by allowing the user to change the counter to be plotted.
3. Even during the scan, the plotted data can be changed in *BLC*, while you have to wait for it to be finished with online.
4. Basic tools for plot analysis are offered with *BLC*.
5. The last main difference is the way that counters are chosen. In online, you can choose them in any moment during the execution. This makes that a new window

is necessary for that fact, as shown in figure 5.

Our idea is that, usually, before starting an experiment the user already knows the devices that are of interest. Although *online* offers more versatility during setup and commissioning, once the setup is finished and known, the user will not need to modify the counters to be plotted and recorded. This is why *BLC* will show a list of motors and counters during its launch, once the user has selected the interesting ones, they won't have to worry about them any more. This is a less flexible option, but this still looks enough for a standard user.

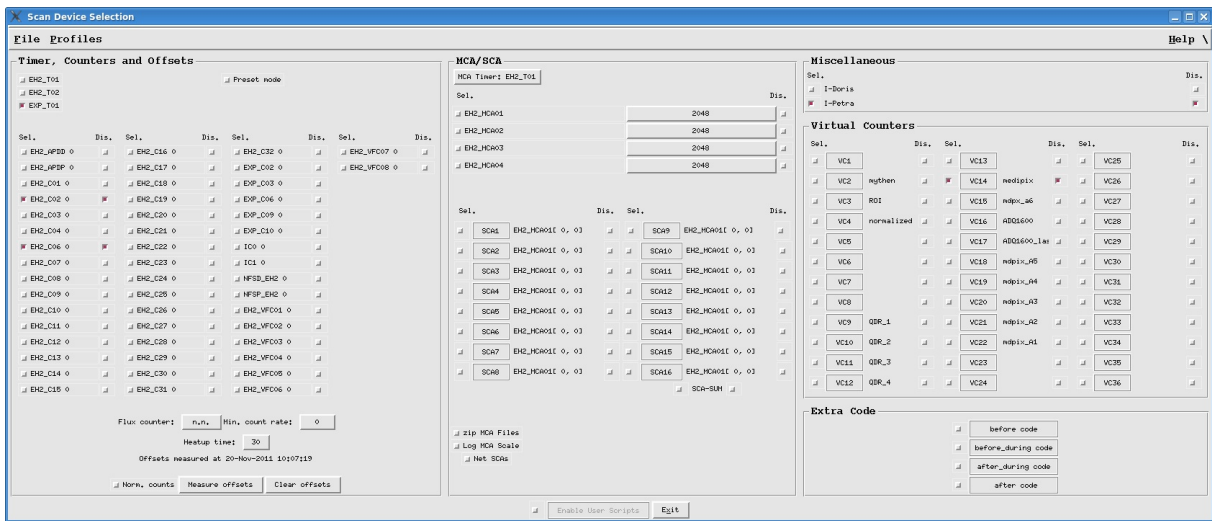


Figure 5: Online - Scan Devices

2. User's Manual

We are showing a brief introduction to the usage, with an explanation of some of the main features supported.

2.1. Launching BLC

BLC is expected to be run on a linux environment that can access to the *Tango Server*. The easiest way to launch the program is with the command line. If you are currently in the folder containing the files related with the program, to launch it you should just enter,

```
python BLC.py
```

Once you run this program, you will get a wizard window that will help you to set general parameters related with the experiment.

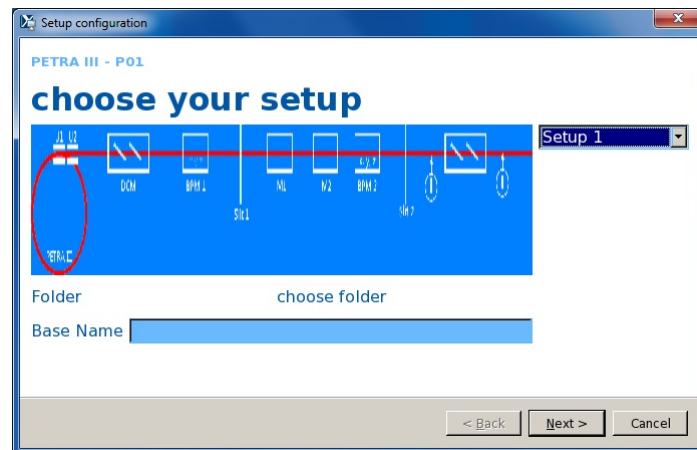


Figure 6: First step in the setup wizard

First, you will have to choose the experimental setup you want to use, as well as the folder and basename for the files created during scans by the program. Note that the files created by *BLC* will be named as `basename_numFile.fio`.

Warning: If you do not give this information properly, no information from the scans will be saved, and sudden closes could happen during the execution.

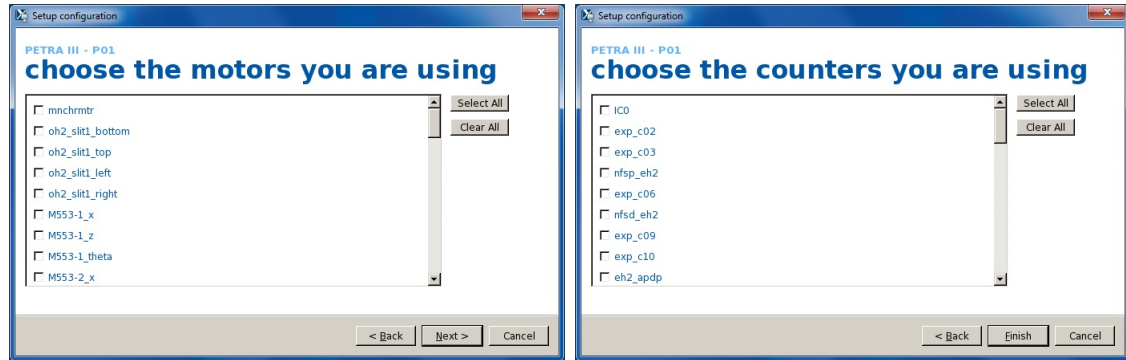


Figure 7: Second and third steps in the setup wizard

In both of the next steps you will have to select motors and counters of interest in your experiment. The motors and counters you choose here are the ones that will be recorded during the scan. This is the only moment where this information can be set.

All the devices offered are taken from the file `online.xml` as long as it can be accessed. If for any reason this file cannot be reached, a local copy is provided where the data will be read from. Anyway, this file may be outdated and some recent changed may not be shown.

2.2. Main window

Once you have configured the general settings for your experiment, a new window will be shown. This is shown in figure 8.

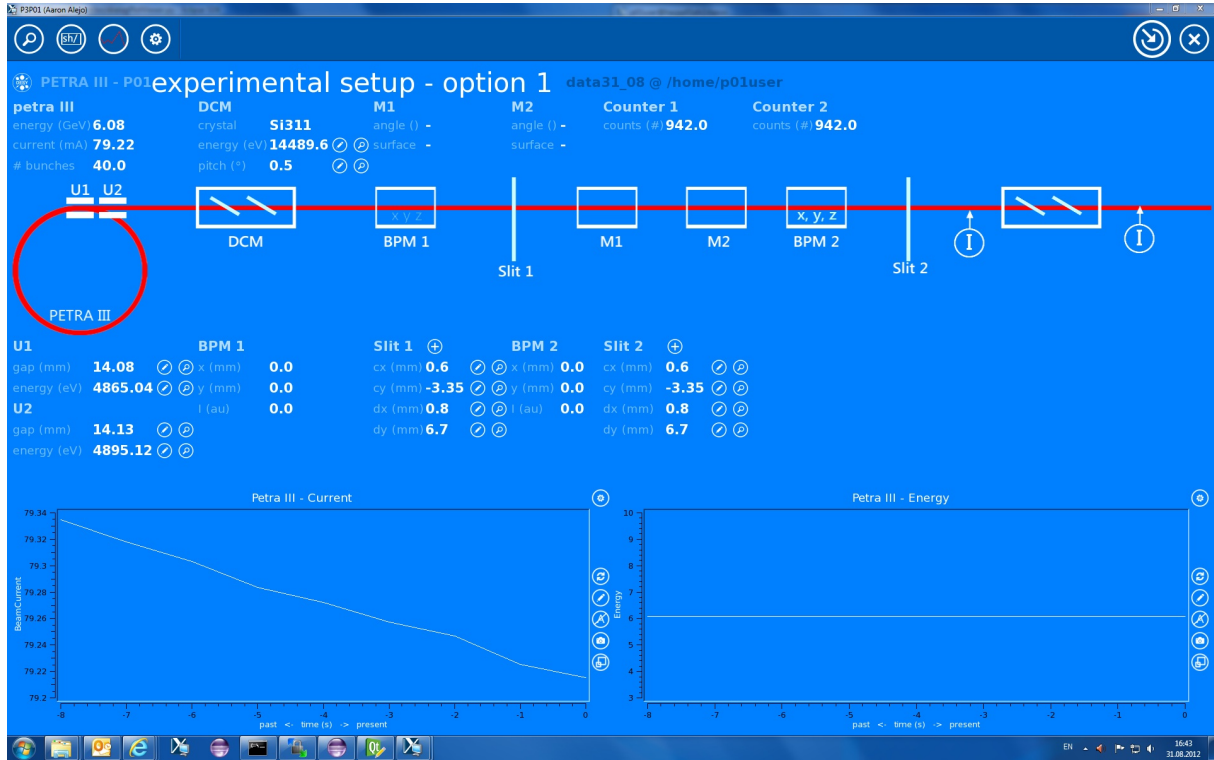


Figure 8: ScreenShot of the Main Window

1. General Scan

Choose this option if you want to scan any of the motors you chose during the launch of the program.

2. Run script

BLC includes its own script executer, so you don't have to go back to the command line.

3. Plot Viewer

Show in one single plot graphs with data taken from files saved from previous scans.

4. Setup settings

Sometimes you are not interested in some parts elements related with the experimental setup. To keep from changing anything related of them, you can disable them with this option. When clicked, you will find a new menu next to the *experimental setup* title, as shown in figure 9.



Figure 9: Setup configuration

You just have to click again in the same button, converted now into a *tick* icon, to confirm all the changes made.

5. Fullscreen mode

BLC is prepared to work in a fullscreen mode. To switch between both view modes you can click this button or use the shortcut *ctrl+F*.

6. Setup title

This label is used as a reminder of the experimental setup you are working with. It cannot be changed once the program has been launched, so you will have to close it and reopen if you want to use a different one.

7. Workplace

This label indicates both the path and the base name for the files created by the program. This means that every scan file will be saved there with the format *basename_number.fio*. For more information about the files go to section

8. Devices info

In the main area of the application, the current values for the most important

attributes of the involved devices are shown. They are updated automatically periodically. Some of the devices show two icons: with these icons you can change the value of that attribute or do a scan of them. Both of these options will be shown later.

9. Plots

Finally, at the bottom of the Main Window, you have plots that allow you to live-plot any attribute of any device in the laboratory. As there are some options related with this plots, we will explain them in its own section.

2.3. Real-Time Plots

In the main window there are three different plots that are updated on real time. This means that you can see three different attributes plotted at the same time. In the next points we are showing how to modify and interact with these plots.

2.3.1. The plot area

In figure 10 the plotting area of the main window is shown.

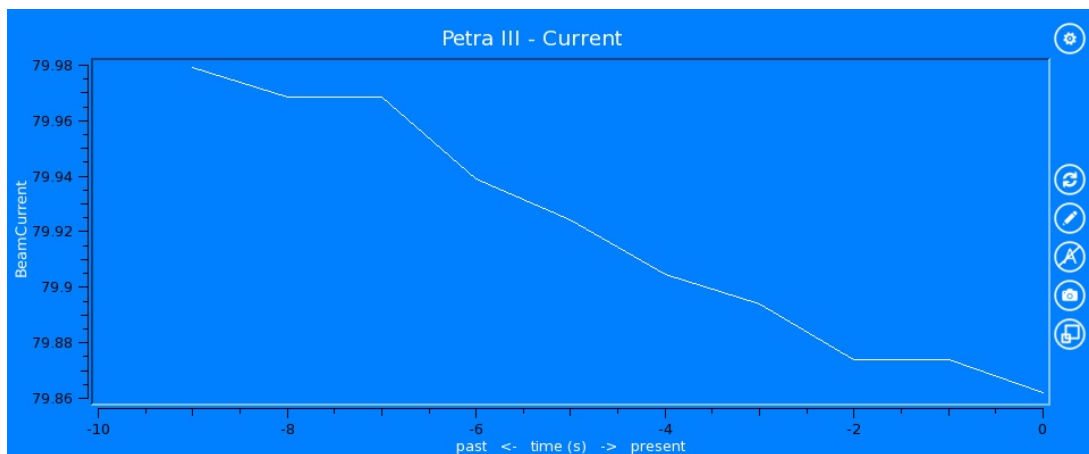


Figure 10: Edit icon

About the plot itself, we can see

- Title, which tells the device and attribute that are actually plotted.
- Y-Axis Label corresponds with the attribute name in the Tango System.
- X-Axis Label corresponds with the time. This time must be understood as the seconds ago since the value was taken.

It is important to note that the maximum time that can be stored in the application is 24 hours. In fact this time may even be too large for some computers. To control this maximum time, you can modify the configuration file, but be careful on changing this parameter. For more information, check appendix related with configuration file.

Also, on the right side of the area you can find some icons to interact with the plot.

2.3.2. Change attribute plotted

To change the attribute currently plotted, you have to click on the corresponding button, which is shown in figure 11.



Figure 11: Button icon

Once you have clicked on this button, a new window will be open. At this point, you will see a set of preconfigured devices and attributes that can be plotted. If the device you are interested in is not in the list, you can switch to the advanced mode by clicking on the *plus button*.

In the advanced mode, first you will have to write down the address for the device you are

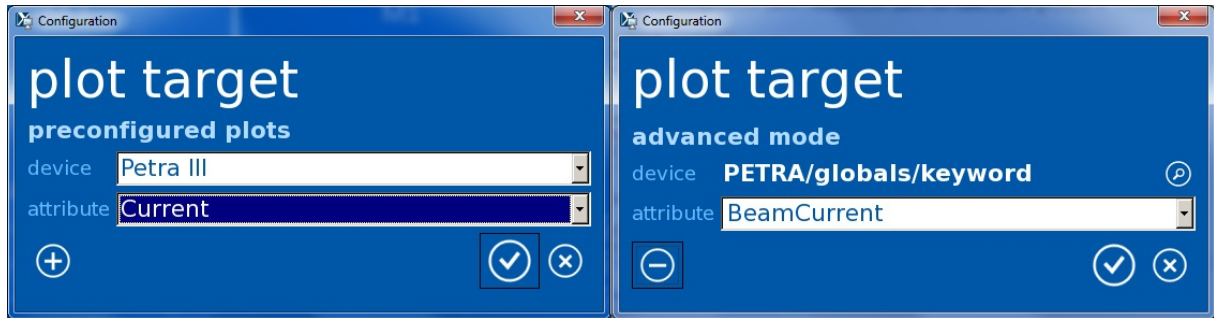


Figure 12: Change the device plotted. On the right side the basic version is shown, with only some devices and attributes offered, while on the left side the *advanced* version is shown, allowing the use of any device

using. You can get the address from *jive* if you do not know it. To load the attributes, you can click on the *find icon*, or click on the attributes list. If the address written does not correspond to an actual device, a message will be shown to warn you.

2.3.3. Modify plot

Some of the properties of the plot itself can be modify. To open the corresponding dialog you can click on the button shown in figure 13.



Figure 13: Edit plot properties icon

Upon clicking on it, you will get a new dialog like the one shown in figure 14.

In the new window, there are mainly two aspects you can modify.

- Change y-axis. You can set the maximum and minimum value plotted. This will only take effect as long as the auto-update is not activated.
- Change x-axis: time. You can choose the time plotted and the update interval. This is the time between two consecutive accesses to the device.

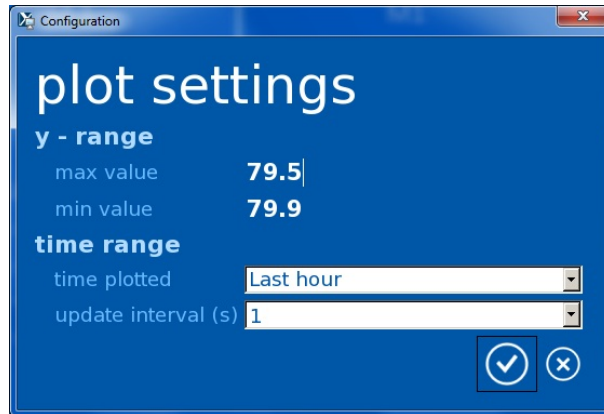


Figure 14: Plot properties modification dialog

2.3.4. Autoupdate axis / Refresh

As we saw in the previous section, you can set the maximum/minimum values plotted. Anyways, sometimes it is better to use the autoupdate option, which will modify the axis in such a way that every data is fitted properly. If you can activate/deactivate this option, you can just click on the corresponding button, shown in figure 15.



Figure 15: Activate/Deactivate autoupdate of axis

If you do not want to use the autoupdate, you can move the axis, so the current value is properly shown on the screen. To do so, just click on the *refresh button* shown in figure 16. Note that this button can only be clicked when autoupdate is deactivated, and it will have no effect in any other case.



Figure 16: Refresh axis

2.3.5. Save Plot

If you want to save the current plot, just click on the *snapshot button*, shown on figure 17. Once you do this, a standard save-dialog will pop up.



Figure 17: Save plot

You can save the plot itself as a new image, or save plain data in a txt file. Just choose the one you want in the save dialog.

2.3.6. Maximize Plot

Generally, the size and colors of the plot as shown in the main window are good enough. However, sometimes a bigger window may be needed. Or an image with a white background might be required. For those cases, you have the possibility of opening the plot in a new window. You can do this by clicking on the *open in new window button*, shown in figure 18.



Figure 18: Open Plot in New Window

2.4. Change an attribute

As it has already been mentioned, you can directly modify the values of some of the attributes from the main window. To do so, you have to click on the edit button corresponding with such attribute. You can see this button in the figure 19.



Figure 19: Edit icon

When clicking on it, as long as the device is connected and working properly, you will get a new dialog like the one shown in figure 20.

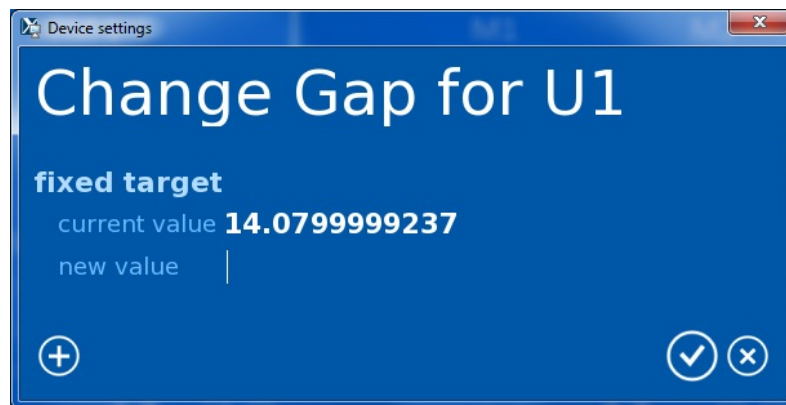


Figure 20: Attribute modification dialog

First thing you can see in the dialog is the title, where both the attribute and the device being modified are shown. Also, a more accurate version of the current value is given to the user. To change it, just write the new value and click the *accept* button.

Note. The *plus* button corresponds to a future advanced version that has not been completely developed, so this button is not enabled yet.

2.5. Scan device

One of the most important things in the software is the scanning. To run a scan, just click on the icon shown in figure 21.



Figure 21: Open scan window

Note that depending on where you choose to open a scan you can have two different windows. If you decide to run a general scan, the window will let you choose any of the motors that you picked up when you run the program. If you choose to scan one of the devices, then you won't be able to change it from the window. In figure 22, the second option is shown.

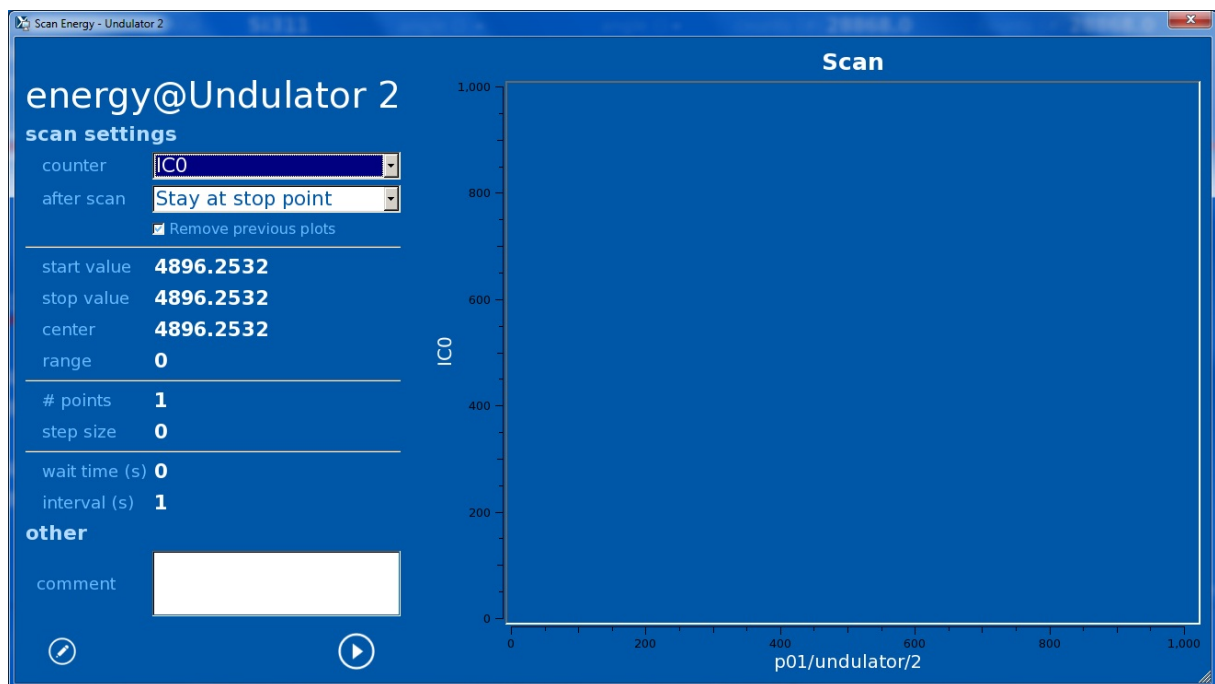


Figure 22: Scan window

Let's see the options in the window

1. **Counter.** Although data for every counter will be saved during the scan, you can choose which one is being plotted. You can switch from one to another any time, even during the scan.
2. **After scan.** Choose what the motor should do after the scan: stay at the stop point, go back to the start point, or to the point where it was before the scan started.
3. **Values.** You can choose between giving a center+range or a start+stop combination. Note that by changing one of them, the rest will be automatically updated to the new values.
4. **Wait time.** Time to wait after the motor finishes moving before starting counting.
5. **Interval.** Time that the counters will be measuring.
6. **Comment.** Text that will be saved with the data in the file.

Once you have configured every option, you can just start the scan by clicking the *play button*. While scanning, you will be able to stop it (which will abort the scan, you will be able to start a new one) or pause it.

2.6. Run script

Run script is a secondary tool that allows you to run any python script, a feedback routine for example, without going back to the shell. To open the window, you should click on the corresponding button, shown in figure 23.



Figure 23: Button to open the run-script window

Once you do this, a new window will be shown. This is shown in figure 24.

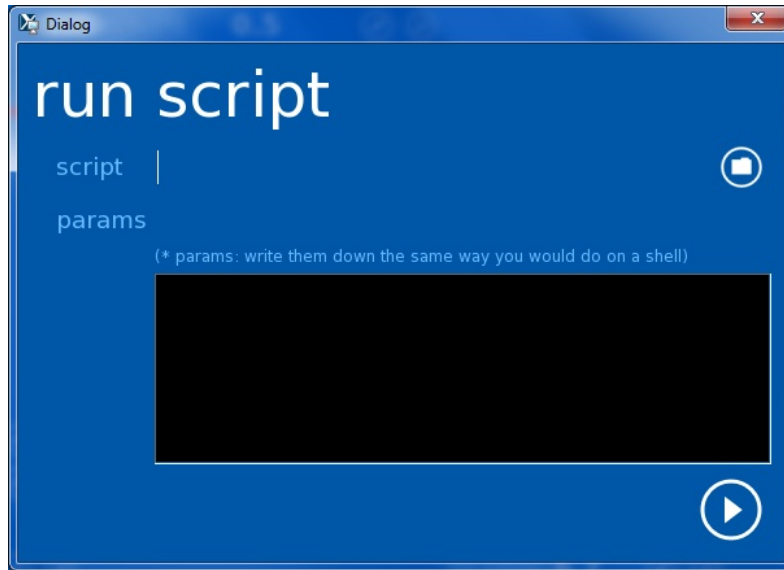


Figure 24: Run script

Let's see the procedure to run a new script

1. Choose your script. You can do it by clicking on the *folder icon*, and selecting it in the window that will appear.
2. Write the params of you script, in case you need them. To do so, just write every parameter as you would do on a shell: everything must be separated with spaces.
3. To run the program, just click on play.
4. The output of your program will be shown in the terminal-like box. **NOTE:** this part is still being tested and some problems with the output may happen, delaying the appearance until the script finishes.
5. The script will continue running until it finishes or until you press the stop button that will appear while running the script.

2.7. Plot Viewer

This feature has been the last one added, and is still being tested. To open the plot viewer just click on the icon in the toolbar, as shown in figure 25.



Figure 25: Button to open the plot viewer

Once you have done this, a new window will be shown. This window is shown in figure 26. As you can see, there are 4 different parts in the window.

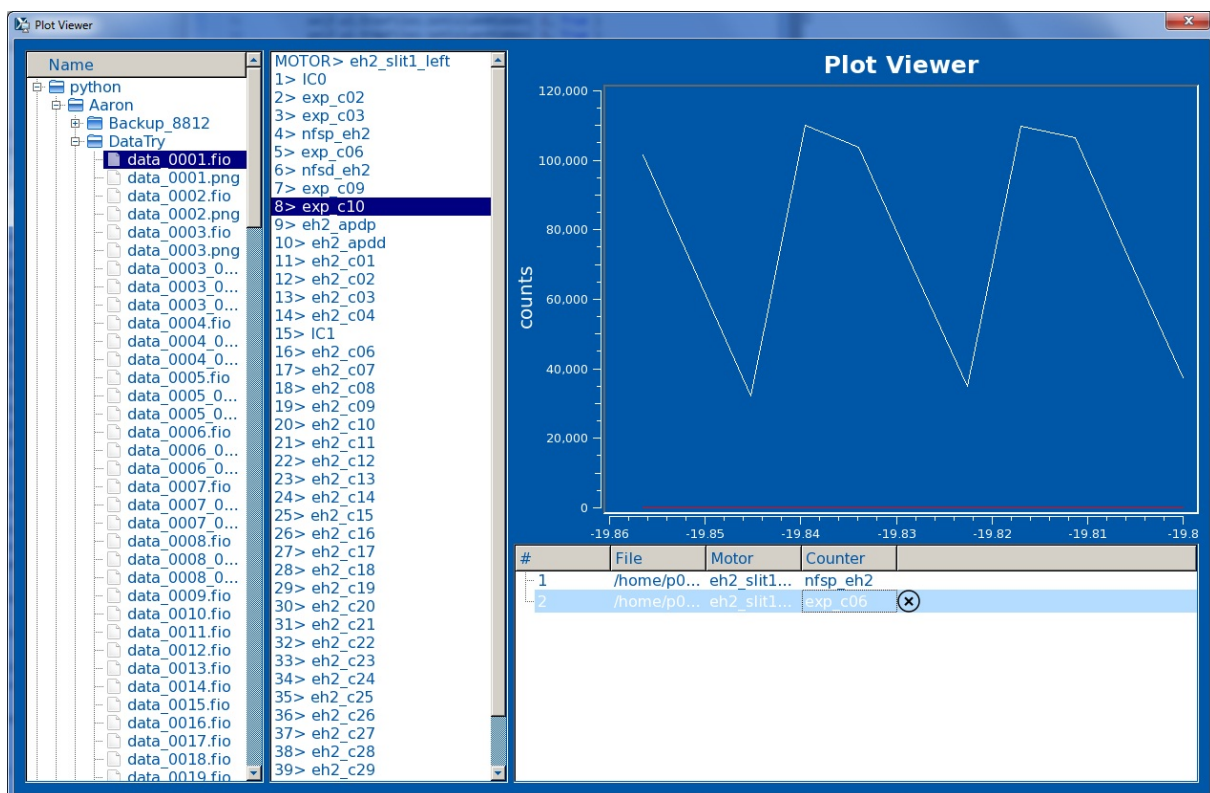


Figure 26: Plot viewer

1. On the left side, you have a file browser. With this you can go to the file including the data you are interested in plotted. Note that, right now, only *fio* files created

with *BLCor online* can be used. Double click on the to show the attributes stored in the file

2. In the center you can see a list of the attributes in the file. The first one is the motor used during the scanning, while the rest are different counters. Double click on the one you want to add to the plotted items list.
3. On the left side, at the bottom, you will find the list of attributes currently plotted. Single click on any of them to highlight it, or double click to remove it from the plot.
4. Finally, on top, you can find all the plots that have been chosen so far.

3. Programmer's Manual

3.1. Introduction

In the following sections, general aspects for the most important classes developed will be described briefly. Also some features concerning the code will be described. For more information, the code is being commented for a better understanding.

3.2. Used Tools

The tools used during the development of the program will be shown following.

1. Eclipse

Eclipse is the main tool used. It has been the programming environment chosen. The flexibility and number of plugins make it one of the most powerful environments right now.

2. PyDev

This is an eclipse plugin that makes programming for python with eclipse a more comfortable target. It offers code-completion and syntax highlighting as some of its main features.

3. QtDesigner

For GUI development, QtDesigner has been used. Among other advantages, we find that it works in different platforms, makes design of GUI really easy and it integrates inside most of the core features of Qt.

3.3. Classes

3.3.1. Device Threads

Introduction The Device Threads are classes derived from *QThread*. The idea behind is to have a different thread running for each of the main devices for the experiment, in such a way that when there is new information to be read, it will be done by the thread, without making the whole software work slower.

Common attributes All the classes created as device threads have most of the attributes in common.

- **Address:** Full path in TANGO device tree for the device accessed.
- **Device:** Probably the most important attribute in the class. This is a Tango object (DeviceProxy) that will let us access to the device.
- **Update Time:** Time between two consecutive accesses to the device.
- **Connected:** Variable indicating if the device was properly connected.
- **Is Used:** In case the device attributes can be modified, the user may have chosen not to allow it during the configuration. This variable shows whether the modification can take place or not.
- **Exiting:** Custom variable used to stop the thread.

General structure All of the threads have a similar structure.

1. During initialization, attributes are set up properly. Device connection is attempted.
2. To start running the thread in parallel execution, `thread.start()` should be called. At this point, a new thread will be actually created, and the code placed in the *run* function will be executed.

3. The thread gets into a loop where it waits for *update time* to read again. Once read, a Qt Signal will be raised to let the rest of the classes that new data is available. This loop will continue until *exiting* is set to *True*.
4. To kill the thread, the method *stop* should be called. When this happens, *exiting* is set to *True*, and the program waits for the thread to finish itself.

Classes related The classes corresponding to *Device Threads* are,

- **AnyDeviceThread:** This is a general purpose class. It can connect to any device and read only one attribute from them. The reason for this class to exist is that during *live plotting*, no limitation on the device chosen is made, so we need a general interface to access all of them. For logical reasons, this class is created as read-only, i.e. no modification of device attributes is allowed.
- **PetraThread:** This class reads data related with the PETRA III storage ring. This is the reason why the class does not offer the possibility of modifying parameters. The attributes saved in the class every time the device is accessed are,
 - Current
 - Number of bunches
 - Energy
- **UndulatorThread:** This class reads data related with the undulators of P01. Also, it offers the possibility of modifying values for the attributes controlled. The attributes saved in the class when device is accessed are,
 - Gap
 - Energy

- **DCMThread:** This class reads data related with the Double Crystal Monochromator. Only some of the attributes can be modified. **Crystal** cannot be changed for security reasons, so for modifications other resources will be necessary. The attributes saved in the class when device is accessed are,
 - Crystal
 - Energy
 - Pitch

- **BPMThread:** This class reads data related with the Beam Position Monitor. None of the attributes can be modified. The attributes saved in the class when device is accessed are,
 - X position
 - Y position
 - Intensity

- **SlitThread:** This class reads data related with the Slits. All of the attributes can be modified. This class allows two working modes. The first one is oriented to used the slit as a whole, choosing center and width. On the other hand, the so called *advanced mode* lets the user modify directly the motors controlling the individual blades of the slit. The attributes saved in the class when device is accessed are,
 - Center - X position
 - Center - Y position
 - Width
 - Height
 - Top motor position
 - Bottom motor position

- Left motor position
- Right motor position
- **CounterThread**: This class reads data from the counters. For obvious reasons, no attribute can be modified with this class. The data saved in the class when device is accessed is,
 - Number of counts

3.3.2. Interface Classes

Although it is not completely true that a *Model-View-Controller* paradigm has been used, it is a fact that a similar approach has been. If we take this into account, this interface classes correspond with the *view*.

All the classes that have only information related with the GUI are named as follows,

$$ui + type + name .py$$

Where type can be window, dialog, wizard...

This files should not be modified, as they have been created with *QtDesigner*. To modify the interface, it would be better to use the designer, and open the corresponding *.ui* file (same name, without *ui* at the beginning). To get the new python file for the GUI, you can use this command,

$$pyuic4 uiFile.ui -o pythonFile.py$$

3.3.3. Model classes

Introduction Here we will describe briefly the classes behind the GUI that were explained in the previous section. Due to the big differences between them, only a few details on their main characteristics will be given.

1. Setup1MainWindow

This class acts as the model for the Main Window for the application when the setup 1 is chosen. It centralizes every functionality in one place, acting as a mediator between the rest of the classes.

The two main activities developed by this class are taking real-time data from the devices, and take care of the real-time plotting of any device chosen by the user.

As mentioned in the corresponding section, the threads will emit a signal every time they have new data read from a device. Using the Qt Signal-Slot system, those signals are connected to functions in the Main Window class in such a way that every time that signal is generated, the main window will read the attributes from that class and show them in the right place.

About the live plotting we will explain later in the *other classes* section.

2. dialogScan

This class is the model for the Scan Dialog Window. It is responsible for the behaviour of both working modes: device scanning and data analysis. The switch between both of them is based on a private variable which determines the current mode, than can be changed by clicking a button.

In the scanning we find the possibility of choosing the counter that is being currently plotted. The list of counters offered is the one the user chose while launching the program, and is set when the dialog-object is created.

When setting one of the values out of maximum/minimum + center/range for the scanning, the rest of them are automatically updated. To do this, we are using the signal *textEdited* (*const QString&*) offered by the *QTextEdit* class. The reason to do this is not to make the user confused, as all of the options are shown simultaneously on screen.

When the *play* button is clicked, a new *ScanThread* object is created with the

parameters entered. This class is explained in the *Other Classes* section. While the scanning is running it can be stop in two different ways:

- a) **Stop.** By clicking this, the scan is aborted. To do this, the thread is going to stop the motor while moving and close the file.
- b) **Pause.** By clicking this, the scan is just paused. This means that you still cannot start a new scan, as the current one is not yet finished. By clicking *play* again, it will continue from the same point it was paused. Note that with this option, the pause won't take place until the current measurement is finished (moving motor + counting for the whole interval).

Finally, we also have in this class the analysis. When switching to this mode, the maximum of the current plot will be calculated to be shown in the proper spot. Then, we have 4 main possibilities to interact with the plot. **Note:** this changes are always made in a new copy of the data stored internally, but the real values measured during the scan will not be modified.

- a) **Derivative.** This will calculate the derivative of the current plot. As we are working with a discrete number of points, the way to calculate the derivative is by taking the ratio of the differences,

$$y'_n = \frac{y_{n+1} - y_n}{x_{n+1} - x_n}$$

$$x'_n = \frac{x_{n+1} - x_n}{2}$$

- b) **Flip Horizontally.** This will flip the plot. To do it, if N is the number of points in the plot, what we do is,

$$temp = y_n$$

$$y_n = y_{N-n}$$

$$y_{N-n} = temp$$

With this, the values that were at the biggest x are now at the lowest, and vice versa.

- c) **Flip vertically.** It will flip the plot vertically around the center value. This means that the maximum will be placed at the y value where the minimum was before, and vice versa. To do so,

$$mean = max(y) - min(y)$$

$$y = mean - y$$

- d) **Fit to Gaussian.** It will fit the data into a gaussian. This is not a tested feature, and only a non-accurate fitting is made, and will be shown on screen with a red curve. The way this fit is made with a formula, not an iterative method. Better fitting is planned.

3. dialogScript

This class is the model for the Run Script Dialog. First, we will show the main **attributes**.

- **fileName:** Path to the file chosen which is the python script to run.
- **params:** Parameters that will be used with the script. It will contain an array of strings.
- **process:** QProcess object in charge of running script in a new thread.
- **isRunning:** internal variable that tells whether a script is being run currently or not.

The operation of this class is mainly based in the signal-slot model that Qt brings. This is why we will explain the slots used in this class.

- **chooseFile:** This slot is connected to the *Open file* button in the GUI. When called, this function will create a new dialog to let the user choose a new file. If the user is already running a script, it will display an alert, and no dialog will be shown. If a file is chosen, the name will be stored in the attribute *filename*.
- **switchMode:** This slot is connected to the *play/stop* button. When clicked, it will check the value of the *isRunning* attribute. If it is set to False, it will start a new process with the file and parameters chosen. If it is set to true, it will stop the process by sending a kill signal.
- **readOutput/readErrors:** These slots are connected to the stdout/stderr signals in the processes. Whenever there is something to be shown by the script, this functions will be called. They will read that output and show it on screen.
- **processExited:** This slot is connected to the signal emitted when process execution is finished. This will change the *isRunning* attribute to the right value, and will show on screen the exit value for the process that just ended.

4. dialogPlotViewer

This class is the model for the Run Script Dialog. First, we will show the main **attributes**.

- **model:** QFileSystemModel object that will access and load the file system. This is where the data shown in the tree is taken from, and where the chosen file will be read from.
- **listColumns :** list of the columns that have been read from the file.
- **treePlotted :** list of the plots that are being plotted currently.

- **xData, yData** : list of arrays with data for each plot plotted.
- **fileName** : path to the file currently selected.
- **numColumns** : number of columns in a given file.

Again, the operation of this class is mainly based in the signal-slot model that Qt brings. This is why we will explain the slots used in this class.

- **fileSelected**: This slot is connected to the double click signal emitted by the *model* whenever a file is chosen. It will check if the file selected is a *.fio* file, and if it is, it will read the columns that are saved in it and will keep show them in *listColumns*
- **columnSelected**: This slot is connected to the double click signal emitted by the *listColumns* whenever a counter is chosen. It will read the data from the file and will add it to the list of elements plotted.
- **markPlot**: whenever a plot is clicked on the list of plots it will be highlighted in red with this function.
- **deletePlot**: when a plot is double-clicked on the list of plots it will be taken out of the list and of the array with the plotted data. Then, the rest of the data will be replotted.

3.3.4. Other classes

Scan Thread This class derives directly from *QThread*. Its goal is to connect with the motors and counters, move the motor being scanned, read data from counters and save that into a file.

Here we show some of the most important attributes of this class

- **motorName, motorDevice, motorAttribute**: Information about the motor that is going to be moved for the scan.

- `startValue`, `stopValue`, `stepSize`: It keeps the information about the range that is going to be scanned.
- `waitTime`: Time in seconds after the motor moves before starting counting.
- `startPosition`: Position of the motor before the scan started. This value is read to allow the user choose to move back to original position after the scan is finished.
- `motors`: list of motors, including name, address and `DeviceProxy` object associated.
- `counters`: the same as `motors`, but with the information about the counters that will be read during the scan.
- `isScanning`, `pause`: private attributes to know if the scan has started or if it is paused

Finally, we are going to explain briefly the most important methods in this class.

- **Run.** This is the method called when the thread starts running. Here we show a scheme of the actions done.
 1. It creates a new scan file, calling the proper function, that will be explained later.
 2. Reads the original position and saves it in the object.
 3. Move motor to start position
 4. Sleep *waitTime* seconds before starting counting.
 5. Start counter
 6. Read from every counter
 7. Save data in file
 8. Move *stepSize* the motor
 9. If `stopValue` is still not reached, go back to sleeping. Else, close scan file and finish.

- **createScanFile.** It will create a new file using the path and basename chosen by the user, adding a number at the end. The structure of this file can be seen on the appendix.

A. FIO File Format

The file format used to save the data from scans is a version of *fio* file format. There are only a few changes made, as it will be shown. First, here we can see an example of basic file.

```
!  
  
! Comments :  
This is a comment  
  
%c  
eh2_slit1_left (p01/motor/eh2.01) -Scan started at 18:26:35.16476 - Scan finished at  
19:08:19.443000  
name: p01/motor/eh2.01 from -19.8 to -19.86 sampling 1.0s, waiting 0s  
no offset corrections due to program version  
  
!  
  
! Parameter  
%p  
mnchrmttr = 14489.6004327  
oh2_slit1_bottom = -0.25  
oh2_slit1_top = 0.25  
  
!  
  
! Data  
%d  
Col 1 : eh2_slit1_left  
Col 2 : IC0  
Col 3 : exp_c02  
Col 4 : exp_c03  
Col 5 : nfsp_eh2  
-19.800000 1133.000000 1138.000000 0.000000 -450928354.000000  
...
```

Let's see the structure of the file,

1. We have the comments that the user decided to save in the file. This is one of the biggest **differences** with the previous *fio* format, which did not allow the use of comments from the user.

2. After the %c, we find the standard comments for fio file. These include motor scanned, start and finish time. All the general information about the scan is saved here.
3. Then we have the parameters, that will be saved right after the %p. In this section, the values of the position for the motors that are interesting for the in the experiment are saved. As only one is going to move, this will be the same for the whole experiment.
4. Finally we have the data itself. This is saved after %d. First thing shown is the meaning for each column. Note that : appear here. This is **important**, and it is a difference with previous fio format. The reason for it to be important is that the plot viewer included with *BLC* will look for them to know where the name of the column starts, and those are not found, then no column will be read. After that, the pure data is stored, with a minimum separation of 1 space between two consecutive columns.

B. Config File

In this file, The information related with the configuration is stored. First, let's see part of the file

```
[online]
path = /online_dir/online.xml

[live_plotting]
max_time = 43200

[address]
petra = PETRA/globals/keyword
undulator_1 = p01/undulator/1
```

As we can see, the structure of the file is really simple. In brackets we have the section which the attributes coming are referring to. Then we have attributes, and their values. The advantage of such a simple structure is that any new configuration necessary can be easily added.

References

- [1] González Duque, R., *Python para todos*, <http://mundogeek.net/tutorial-python>
- [2] Qt Developer Network, *Qt Documentation Pages*, <http://qt-project.org/doc/qt-4.8>
- [3] *Tango Documentation*, <http://www.tango-controls.org/Documents>
- [4] Hasylab, *Tango at Hasylab*, <http://hasyweb.desy.de/services/computing/Tango/Tango.pdf>