ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THPV014

ADOPTING PyQt FOR BEAM INSTRUMENTATION GUI DEVELOPMENT AT CERN

S. Zanzottera, S. Jensen, S. Jackson, CERN, Geneva, Switzerland

Abstract

As Java GUI toolkits become deprecated, the Beam Instrumentation (BI) group at CERN has investigated alternatives and selected PyQt as one of the suitable technologies for future GUIs, in accordance with the paper presented at ICALEPCS19.

This paper presents tools created, or adapted, to seamlessly integrate future PyQt GUI development alongside current Java oriented workflows and the controls environment. This includes (a) creating a project template and a GUI management tool to ease and standardize our development process, (b) rewriting our previously Java-centric Expert GUI Launcher to be language-agnostic and (c) porting a selection of operational GUIs from Java to PyQt, to test the feasibility of the development process and identify bottlenecks.

To conclude, the challenges we anticipate for the BI GUI developer community in adopting this new technology are also discussed.

INTRODUCTION

The software section of the Beam Instrumentation Group at CERN (SY-BI-SW) has a mandate, to provide expert GUIs allowing hardware experts to manage and diagnose instrumentation. As explained in detail in our preliminary evaluation[1], the software stack consists of several layers, where the most high-level ones, including the GUIs, have been traditionally implemented in Java. However, as Java GUI technologies age and become deprecated, efforts[1,2,3] were made to identify suitable, more modern replacements. As these evaluations concluded, no alternative Java framework could be identified, leading to the option of PyQt.

Adopting PyQt is not trivial: Firstly, tools, services and frameworks must be developed for integration with CERN's control system. Secondly, existing GUIs cannot simply be migrated – they must be rewritten. This represents a massive effort in terms of redesigning and reprogramming. In addition, developers will have to adopt Python as a programming language, which is a challenge in itself, as Python is fundamentally different from Java in many aspects.

As integration with CERN’s control system is addressed by another team at CERN, we have been able to focus on the GUI programming aspect itself (GUI management tools, widgets, proof of concept GUIs) and also on the adaptation of our decade-old Java-oriented GUI development workflow to a more language-agnostic one, supported by more generic tools.

PYTHON TOOLS FOR EXPERT GUIS

Devtools: bipy-gui-manager

Previous integration efforts resulted in a set of tools and environments under the name of Acc-Py[4]. However, none of these tools are oriented towards GUI development: they all target a generalised Python codebase, favouring in practice CLI applications and libraries. Consequently, we proceeded to define a set of best-practices to be followed in order to develop Expert GUIs with PyQt and embarked on the creation of a specific tool, the “bipy-gui-manager” to encourage (and partially enforce) them.

Upon invocation, this command line utility collects some basic project information (project name, author name, author email etc…) and then creates a template project in the desired location, pre-configured with its own GitLab repository (created on the fly), a dedicated virtual environment, a template for Sphinx-based documentation and a workflow that provides Continuous Integration, Continuous Deployment and Continuous Documentation for the project. As a consequence, the setup effort required from the developer to get a fully standard project is close to zero. Even the README is pre-written by compiling a template README with the information gathered by the tool at setup time.

The bipy-gui-manager enables us to enforce group-specific conventions and promote best practices in general. This is valuable in homogenizing the code produced, given the number of short-term developers in the section and their different backgrounds.

One example is how bipy-gui-manager deals with GitLab repositories. In the past, the section had problems with critical pieces of software not checked into version control, or not having their repositories synchronized with the code that was effectively in production. The bipy-gui-manager addresses the issue by 1) setting up the repository for the developers, so even if they don’t know or have no time for version control, the tool takes care of it, and 2) by not allowing the developers to use its simplified release function unless they commit all their changes to GitLab. It is important here to note that bipy-gui-manager does not really block the developer from releasing uncommitted code: a slightly more expert person can still do a release in a single (although longer) command. However, we believe that such small hurdles will make programmers follow conventions, which in turn will help lower our code hand-over and maintenance efforts. This is especially true for projects made by newcomers and interns, who are often tasked with developing or maintaining expert GUIs as a way to familiarize themselves with systems they will eventually be working on.

The bipy-gui-manager is already in use in the section for our first Python Expert GUIs.

RAD Tools: ComRAD

Along with the development of Acc-Py, the team produced another interesting tool for Rapid Application Development (RAD), called ComRAD[5]. This paper will not go into detail about the implementation details of this zero-code GUI tool, but rather highlight how it has contributed to our PyQt efforts.

While initially aimed only at prototypes and fixed-display (non interactive) applications, its scope has been gradually expanded, and it has eventually became an interesting tool for general Expert GUI development.

Therefore, we decided to perform an evaluation of this tool and understand which role it could take in our standard PyQt development workflow. Our requirements for a successful evaluation of this tool included:

- Do not limit the developers from using the full capabilities of PyQt. This is important to allow Expert GUIs to grow in complexity if the need arises.
- Allow the use of custom widgets, to be able to extract components and reduce the development time.
- Be able to package and deploy ComRAD applications on our NFS file system as a regular Python application.
- Have some actual benefit over bare PyQt. This was important to avoid adding unnecessary complexity to our tools without any effective gain.

Our analysis was generally positive: ComRAD matched all our requirements and exceeded them by cutting development time for simple Expert GUIs to hours rather than days.

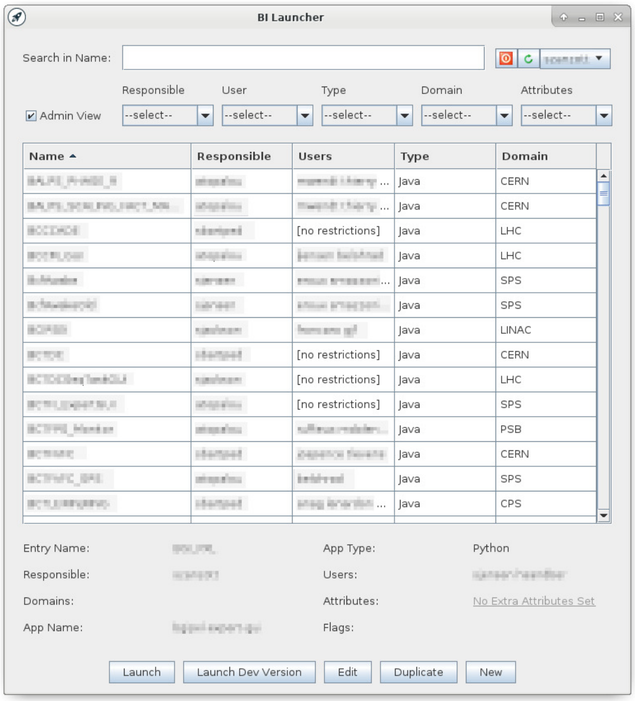
Consequently, ComRAD was added to our workflow. Such an addition was performed by adapting bipy-gui-manager, which is now capable of creating both PyQt and ComRAD oriented Expert GUI projects.

This operation also reinforced the relevance of bipy-gui-manager for controlling our workflow: the tool was easy to adapt to the new standard, and enabled all our developers to develop ComRAD based applications almost immediately at the end of our evaluation and the decision to make it available to developers.

A LANGUAGE-AGNOSTIC LAUNCHER

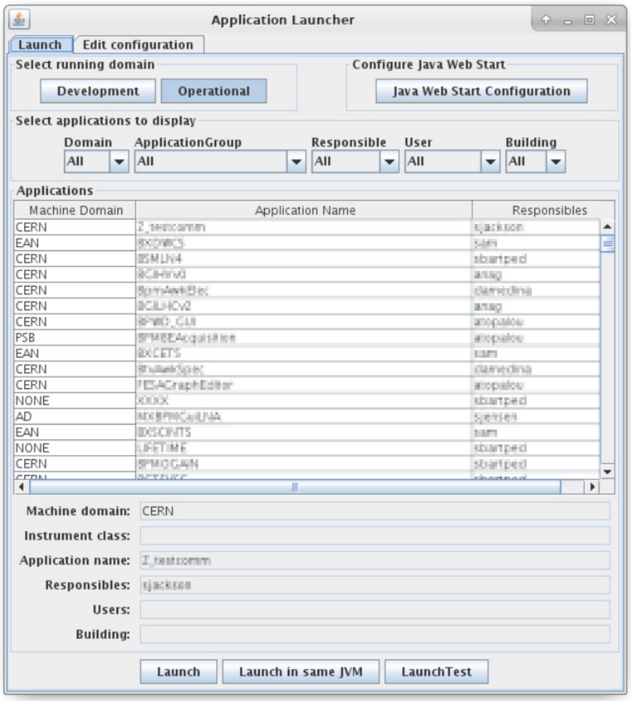
One pillar of our Expert GUI ecosystem is the AppLauncher[6] (Figure 1), a tool which provides two main functionalities: a) it lets us manage user access to expert GUIs, and b) it provides users a centralized catalogue of the expert GUIs a user has access to.

On the operational computers in CERN's Control Center, the console managers point not to the application's binary itself, but to the AppLauncher executable, with a parameter set to the desired application name. This layer of abstraction allows on one side the operators to have a reliable entry point to find applications, and on the other hand gives the developers more freedom on where and how to install their applications for operational use, so long as they can be registered in the AppLauncher and be launched on the target machine.



Name	Responsible	Users	Type	Domain
[illegible]	[illegible]	[illegible]	Java	CERN
[illegible]	[illegible]	[illegible]	Java	CERN
[illegible]	[illegible]	[no restrictions]	Java	LHC
[illegible]	[illegible]	[illegible]	Java	LHC
[illegible]	[illegible]	[illegible]	Java	SPS
[illegible]	[illegible]	[illegible]	Java	SPS
[illegible]	[illegible]	[illegible]	Java	LINAC
[illegible]	[illegible]	[no restrictions]	Java	CERN
[illegible]	[illegible]	[no restrictions]	Java	LHC
[illegible]	[illegible]	[no restrictions]	Java	SPS
[illegible]	[illegible]	[illegible]	Java	PSB
[illegible]	[illegible]	[illegible]	Java	CERN
[illegible]	[illegible]	[illegible]	Java	SPS
[illegible]	[illegible]	[illegible]	Java	CPS

Entry Name: [illegible]
App Type: Python
Responsible: [illegible]
Users: [illegible]
Domains:
Attributes: No Extra Attributes Set
App Name: [illegible]
Flags:



Domain: All
ApplicationGroup: All
Responsible: All
User: All
Building: All

Applications

Machine Domain	Application Name	Responsibles
CERN	[illegible]	[illegible]
EAN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
PSB	[illegible]	[illegible]
EAN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
CERN	[illegible]	[illegible]
NONE	[illegible]	[illegible]
AD	[illegible]	[illegible]
EAN	[illegible]	[illegible]
NONE	[illegible]	[illegible]
CERN	[illegible]	[illegible]

Machine domain: CERN
Instrument class:
Application name: [illegible]
Responsibles: [illegible]
Users:
Building:

Figure 1: The BI Launcher (left) and the old AppLauncher (right).

Given the nature of expert GUIs, accidental misuse could lead to serious equipment damage; however, the AppLauncher did not enforce any strict access control, and it was rather oriented towards helping users finds the correct GUI faster access rather than limiting access to them. In addition, being designed to support Java applications only, it was relying at its core on the Java Web Start system, which has been preserved at CERN after its demise by Oracle as an in-house tool called JWS. The AppLauncher, although rather simple, could not be repurposed to support any other language, and therefore this option was excluded in favour of a more radical one: a full rewrite. The new AppLauncher, now called simply BI Launcher, is a new Java Swing application that imitates to some degree the look and feel of its predecessor, while fundamentally rethinking its internals. By adopting more modern Java standards and reducing the features to what the section mostly uses this tool for, the BI Launcher ended up as a simpler application with a compact codebase, able to launch Java (though JWS), Python (through Acc-Py tools) and Web pages (through the browser) from the same interface. It also modernised the security aspects of accessing the application catalogue, by utilising CERN's widely used role-based-access system (RBAC[7]).

The old AppLauncher was eventually decommissioned in September.

FIRST PYTHON EXPERT GUIS

PyQt Expert GUIs

In parallel to the work detailed above, we also ported a few Expert GUIs to PyQt to use them as a proof of concept, and to identify potential blocking issues that we might have overlooked. Candidate GUIs for this role had to match the following requirements:

- *Be small:* in order to be able to quickly iterate in case of issues, a good candidate GUIs should be small, both in the expected codebase size and in the interface.
- *Be thin*: we are not interested at this stage in GUIs that contain a lot of logic unrelated to the interface (like complex calculations or domain-specific algorithms). We need GUIs that do nothing more than allowing the user to read, write and monitor values.
- *Already due for a rewrite*: some GUIs were already in need of a full rewrite. For example, we have a number of JavaFX-based GUIs which we want to migrate, due to the fact that JavaFX is no more a recommended technology for GUIs. Other examples include GUIs based on Swing or Qt (C++), developed by other teams but assigned to our section for maintenance.
- *Include commonly used widgets:* expert GUIs tend to include common widgets, like plots, timing displays, log viewers, toggles, etc. Testing the new technology on these components was mandatory.
- *Be actively used*: the ultimate test is operational use. We opted for GUIs being actively used, rather than made-up test cases.

After an inspection of the collection of GUIs that matched the above requirements, we selected a couple of them for the initial port. The chosen applications were:

- A medium-small application already partially ported to PyQt by an intern in a previous evaluation of the technology, which we are going to identify as "BGI App";
- A very small JavaFX application with some interesting features and plot customizations, which we are going to refer to as "BCF App" (Figure 2).

We began the process from the BGI App in an attempt to define a good project template to be followed by other applications. We defined the features at a project level (GitLab repo, test suite, CI, README, code structure, etc) but also at a visual level (mandatory widgets like the log panel at the bottom of the window, timing bar at the top, etc…) and used the resulting template in the bipy-gui-manager to ensure uniformity for future applications.

Once this step was complete, we moved on to the BCF App, the first real port from scratch. In this case we started with the newly created template, and we verified that the bipy-gui-manager was an effective tool for the task. Then we proceeded with the actual port and succeeded in producing a fully functional application which replicated faithfully all the features present in its JavaFX counterpart.

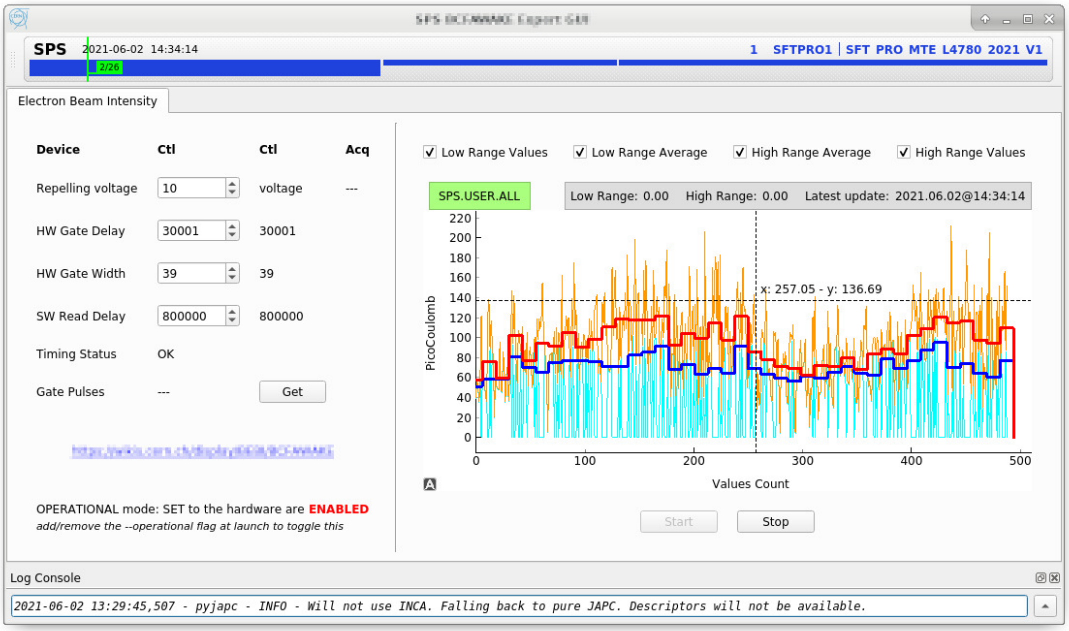
In the process of porting both application we also produced a number of reusable PyQt widgets that, especially once repackaged in BE-CSS widget libraries, are going to make building future PyQt applications even faster and simpler than it was for these ones. Examples of these widgets include the Timing Bar, a widget that displays timing information of a selected accelerator, the Log Console, a highly customizable widget that receives, displays and filters the logs, the crosshair, that was added to the basic plots widget after our example, plus several others that are still being evaluated.

ComRAD Expert GUIs

Shortly after porting the two Expert GUIs mentioned above, ComRAD became ready for evaluation. As a test case, we chose to migrate and simplify a C++ Qt-based application called "PXL App".

Our experiences with ComRAD were considered highly successful, due to a series of benefits that we identified while working on the application:

- *Faster development*. The initial iterations of the GUI were very fast and required barely any code, due to the great amount of Qt UI files that could be reused from the C++ based counterpart. In addition, ComRAD provides useful CERN-specific widgets which facilitated shorter iterations.



Device	Ctl	Ctl	Acq
Repelling voltage	10	voltage	---
HW Gate Delay	30001	30001	
HW Gate Width	39	39	
SW Read Delay	800000	800000	
Timing Status	OK		
Gate Pulses	---		

Low Range: 0.00 High Range: 0.00 Latest update: 2021.06.02@14:34:14

OPERATIONAL mode: SET to the hardware are **ENABLED**
add/remove the --operational flag at launch to toggle this

```
2021-06-02 13:29:45,507 - pyjapc - INFO - Will not use INCA. Falling back to pure JAPC. Descriptors will not be available.
```

Figure 2: Example of an expert GUI ported to PyQt from JavaFX: the BCF App.

- *Decoupling interface design and interface logic.* ComRAD enabled us to design the interface with no boilerplate code to support it, through enhancements made on the Qt Designer. This decoupling allowed us to first build an application with no connections to the middleware and iterate on the design without the need to think about the code.
- *Progressive wiring.* ComRAD allowed us to progressively connect widgets from the simplest to the most complicated. In addition, for the majority of the widgets, it required no code at all to connect to the control system. This code-less approach could be used also in simple plot widgets, and code was required only for the most complex ones.
- *Allows pure PyQt code*. ComRAD was designed to allow developers to fall back to a pure PyQt-style application where needed, while allowing the rest of the application to benefit of its RAD facilities. In fact, while ComRAD has its own enhanced Designer files, it can read and manage regular Qt Designer interfaces too; while it has its own highly automated system to connect to the middleware, it also allows developers to use their own connections, and so on. This is a very strong feature, as it means that the framework will not hinder the developer if they need to implement something different from what ComRAD was designed for, which means that it will be easier to support in the long-term, and even to replace at a future date if the need arises.

Performance Assessment

One of the most worrying concerns of porting our Expert GUIs to Python-based technologies was the performance of some demanding widgets, for example tables and plots. While we met no performance bottlenecks during the development and operation of the first two PyQt GUIs, we finally faced the problem with the PXL App, when we ported it to ComRAD. This app was initially written in C++, and not in Java, due to several reasons: some demanding plots, hundreds of widgets to display, and a special C++ library being used in the backend. Although simplified in its ComRAD version, this GUI still featured a plot that was required to fully re-render up to 1.048.576 points (1024 lines with 1024 data points each) in less than a second in order not to freeze or skip updates.

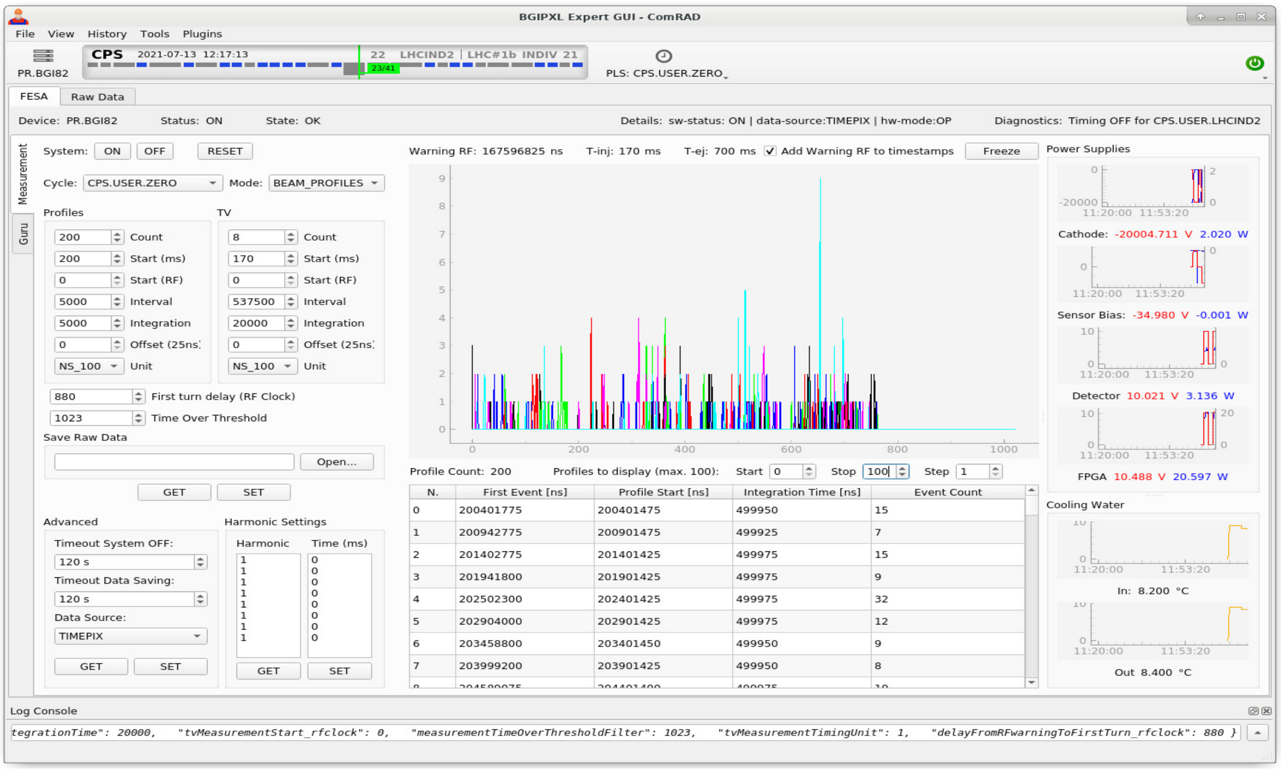
We soon realized that such requirement could not be satisfied, and that the GUI would be able to barely deal with 400 such lines (409.600 points). However, after presenting the issue to the users of this application, we realized that there was no need to display the entire bulk of the data and that displaying a subset of 100 lines was sufficient. With this new concession, the application was able to run smoothly.

FUTURE WORK

Now that the technology is proven and some GUIs have been successfully ported, the next steps involve primarily the port of all the remaining GUIs that need a rewrite. This port will not be simply a translation of the old codebase into a new one: at every iteration we plan to assess which components can be transformed into generic widgets, extract them, package them, and release them to be reused as ComRAD widgets for the following applications.

We believe this process will benefit both ourselves, by speeding up the development of more complex application, and the wider PyQt and ComRAD community at CERN.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THPV014



Device: PR.BGI82 Status: ON State: OK Details: sw-status: ON | data-source:TIMEPIX | hw-mode:OP Diagnostics: Timing OFF for CPS.USER.LHCIND2

System: ON OFF RESET

Cycle: CPS.USER.ZERO Mode: BEAM_PROFILES

Profiles		TV	
200	Count	8	Count
200	Start (ms)	170	Start (ms)
0	Start (RF)	0	Start (RF)
5000	Interval	537500	Interval
5000	Integration	20000	Integration
0	Offset (25ns)	0	Offset (25ns)
NS_100	Unit	NS_100	Unit

880 First turn delay (RF Clock)
1023 Time Over Threshold
Save Raw Data

Advanced
Timeout System OFF: 120 s
Timeout Data Saving: 120 s
Data Source: TIMEPIX

Harmonic Settings

Harmonic	Time (ms)
1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0

Warning RF: 167596825 ns T-inj: 170 ms T-ej: 700 ms ✓ Add Warning RF to timestamps

Profile Count: 200 Profiles to display (max. 100): Start 0 Stop 100 Step 1

N.	First Event [ns]	Profile Start [ns]	Integration Time [ns]	Event Count
0	200401775	200401475	499950	15
1	200942775	200901475	499925	7
2	201402775	201401425	499975	15
3	201941800	201901425	499975	9
4	202502300	202401425	499975	32
5	202904000	202901425	499975	12
6	203458800	203401450	499950	9
7	203999200	203901425	499950	8

Cathode: -20004.711 V 2.020 W
Sensor Bias: -34.980 V -0.001 W
Detector 10.021 V 3.136 W
FPGA 10.488 V 20.597 W
In: 8.200 °C
Out 8.400 °C

```
tegrationTime": 20000,   "tvMeasurementStart_rfclock": 0,   "measurementTimeOverThresholdFilter": 1023,   "tvMeasurementTimingUnit": 1,   "delayFromRFwarningToFirstTurn_rfclock": 880 }
```

Figure 3: Example of an expert GUI ported to ComRAD from C++ Qt: the PXL App.

CONCLUSION

Adopting PyQt as the main framework for Expert GUIs development in our group was not a straightforward task. Most of the assumptions made in the past about the applications had to be re-evaluated to allow non-Java based GUIs to enter the scene. With the help of another team (BE-CSS) which took care of the middleware integrations, we progressively cleared the path for seamless PyQt GUI development by: a) modifying the AppLauncher to enable launching of any type of GUI application, b) developing the bipy-gui-manager to promote standardization and best practices within our team c) porting two pilot GUIs to PyQt, d) porting one GUI to ComRAD and e) assessing the performance capabilities of the plots in a demanding real use-case.

While the future for PyQt as the main technology for future GUIs is still not fully clear, it is clear that PyQt will be a key technology for RAD purposes. In this context, the work put in place to maintain, control and catalogue these efforts will be valuable regardless of whether we decide to move our entire portfolio of GUIs to PyQt technologies or not, and we expect this workflow to serve us well in the years to come.

REFERENCES

[1] S. Bart Pedersen and S. Jackson, "Graphical User Interface programming challenges moving beyond Java Swing and JavaFX", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 637–640. doi:10.18429/JACoW-ICALEPCS2019-MOPHA173

[2] I. Sinkarenko, S. Zanzottera, and V. Baggiolini, "Our journey from Java to PyQt and Web for CERN accelerator control GUIs", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 807-811. doi:10.18429/JACoW-ICALEPCS2019-TUCPR03

[3] S. Zanzottera, "Evaluation of Qt as GUI framework for accelerator controls", M.Sc. thesis, Politecnico di Milano, Italy, 2018.

[4] S. Zanzottera, "Status of Python for GUIs", https://indico.cern.ch/event/1031183/contributions/4330130/attachments/2239943/3797572/Status%20of%20Python%20for%20GUIs.pdf

[5] I. Sinkarenko, "Python GUI Status Update" https://indico.cern.ch/event/1025056/contributions/4303998/attachments/2235763/3789373/Python%20GUI%20Status%20Update.pdf

[6] P. Karlsson and S. Jackson, "The Introduction of hierarchical structure and application security to Java Web Start development", in *Proc. ICALEPCS'05*, Geneva, Oct 2005, paper TH3A.2-5O

[7] K. Kostro, W. Gajewski, and S. Gysin, "Rolebased authorization in equipment access at CERN", in *Proc ICALEPCS'07*, Knoxville, Tennessee, USA, Oct. 2007, paper WPPB08, pp. 415-417.