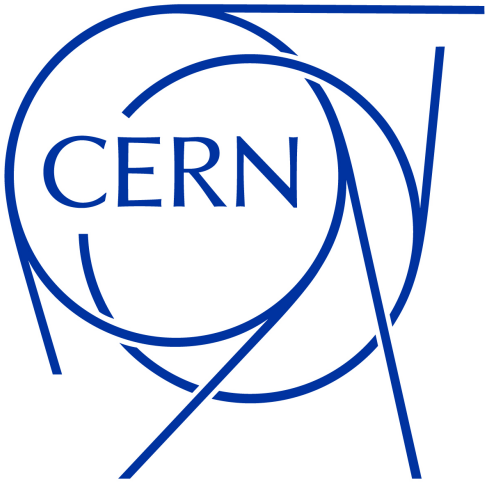

SPS SPILL GUI / BCSPILL SPS GUI

Zübeyde Civelek

Supervisors: Elif Balcı, Stephen Jackson

CERN, Prevessin, France

Abstract

A new beam instrumentation within SPS, the SPS SPILL Monitor, monitors the quality of the spill during extraction processes. During my summer student programme at CERN, I developed the BCSPILLSPS Expert Graphical User Interface(GUI) with Python. The GUI provides an interface to the beam instrumentation experts allowing them to control the settings and visualize the acquired data from SPS SPILL Monitor.

Keywords: BCSPILLSPS, Expert GUI, Graphical User Interface.

Date: 15 September 2023

1 Introduction

During my summer student program, I had the opportunity to work with the Software Section of the Beam Instrumentation (BI) Group of the Accelerator Systems Department (SY-BI-SW).

The responsibilities of the BI Software Section includes developing, testing, diagnosing, and maintaining the software necessary for different instruments produced by the BI Group. [1]

Responsibilities of the SY-BI-SW Section are listed in their website as:

- "Develop Real Time Front End Software for instruments, including the remote control communication interface.
- Develop Expert GUI for instruments.
- Develop tools to allow HW specialists to build their own test programs.
- Take an active part in the commissioning and performance assessment of instruments." [1]

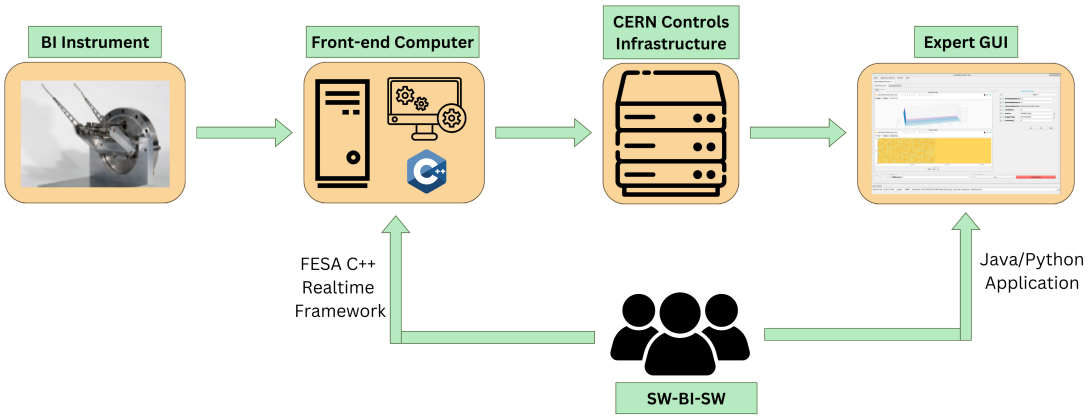

Figure 1: SY-BI-SW Section [1]

My task in the section was to develop a GUI that allows the instrumentation experts to control the settings and visualize the data obtained from the BI instrument.

BI Instrument for SPS SPILL GUI:
The SPS SPILL Monitor (BI Instrument) is a new tool for monitoring spill quality during extraction operations in the SPS.

2 Development Environment/Tools/Libraries

To facilitate interaction with the SPS Spill Monitor, I needed to develop the BCSPILL SPS GUI within a virtual machine (VM). To begin, I established a connection to my VM using TigerVNC and MobaXterm. After that, I started GUI development using Python within the PyCharm IDE.

I used Python libraries such as PyQt and NumPy to develop the GUI. "PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in" [2]. "NumPy is a Python library that provides multidimensional array objects, including mathematical, logical operations, sorting, selecting, discrete Fourier transforms, and random simulations" [3].

I also used CERN libraries such as AccWidgets and ExpertGUICore. AccWidgets (PyQt accelerator widgets) is a collection of common PyQt widgets for CERN accelerator graphical applications [4]. The most crucial library was the ExpertGUICore library developed in my section.

Expert GUI Core

The Expert GUI Core is an internal library designed to simplify GUI application development in Python. It simplifies widget creation and enables easy communication with FESA devices. I also contributed to the development of the Expert GUI Core library by developing a new SimuComm Class.

SimuComm

The SimuComm class is a Python class I developed as part of the ExpertGuiCore library. GUI development often relies on data from FESA devices, which can present challenges when those devices are in use by other developers or processes. Unintentional alterations to the data or settings on the FESA device can lead to undesirable results or interruptions.

All of these factors create a significant dependency on device availability for the development process. However, the SimuComm class provides a solution by enabling the use of simulated data. It effectively mitigates these issues, ensuring a more controlled and dependable environment for GUI development, unaffected by device availability constraints.

The SimuComm class includes methods such as ‘get‘, ‘subscribe‘, and ‘unsubscribe‘ to send data.

There are three important objects in the SimuComm class:

- **SimuDataObject**: Used to send the data you want to simulate to SimuComm.
- **SimuTimerObject**: Used to send data according to the given interval when you want to subscribe.
- **Listener**: Used to determine what to do with sent data.

To get used to these libraries, I first created test projects. I examined template projects. I chose a template and started developing the BCSPILL SPS Expert GUI by integrating SimuComm into my project. I created a Gitlab repository and pushed all the changes in my code there.

Expert GUI Core library provided me with many widgets to visualize the data, interact with the FESA settings and acquisition, and modify the appearance of the GUI.

After the GUI was developed using SimuComm, the functionality of the GUI was tested with the real FESA instrument.

3 BCSPILL SPS GUI Functionalities

The GUI connects to the FESA device developed for SPS SPILL Monitor using the "Expert GUI Core" library.

FESA device: Software representation of a physical BI instrument.

The GUI visualizes the data received from the FESA device using various methods and allows experts to control the settings.

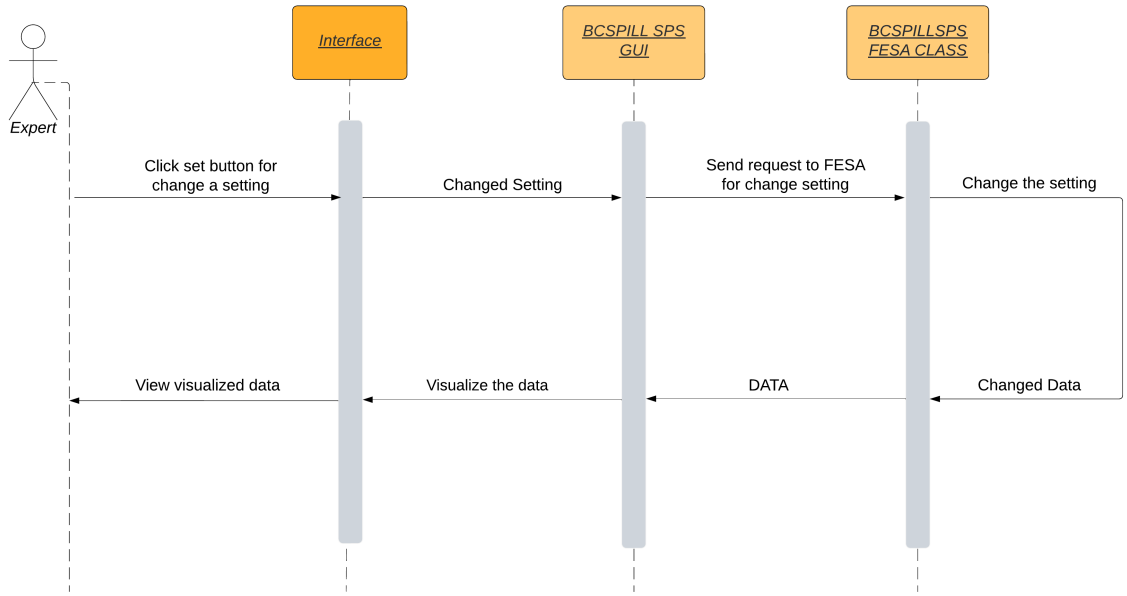

Figure 2: Sequence Diagram

3.1 Design of the BCSPILL SPS GUI

The GUI design is an important element of the project. A good design makes it easier for experts to use the GUI so they can easily interact with the BI instrument.

With a well-designed GUI, complex tasks can be streamlined, it reduces the potential for human error and it ensures safer and more efficient beam control.

In the development of the BCSPILL SPS GUI, attention was given to the following aspects:

User-Centered Design: Feedback was gathered to ensure alignment with the needs and preferences.

Easy Navigation: The development of an easy-to-use navigation was prioritized so that experts could quickly access important data and controls.

Real-Time Visualization: Live data feeds and responsive charts were provided to ensure that experts had access to up-to-date information.

Error Prevention: Validation checks and warnings were implemented to minimize the occurrence of mistakes.

Consistency: Consistency in design elements, such as color schemes and button placement, was maintained for improved usability.

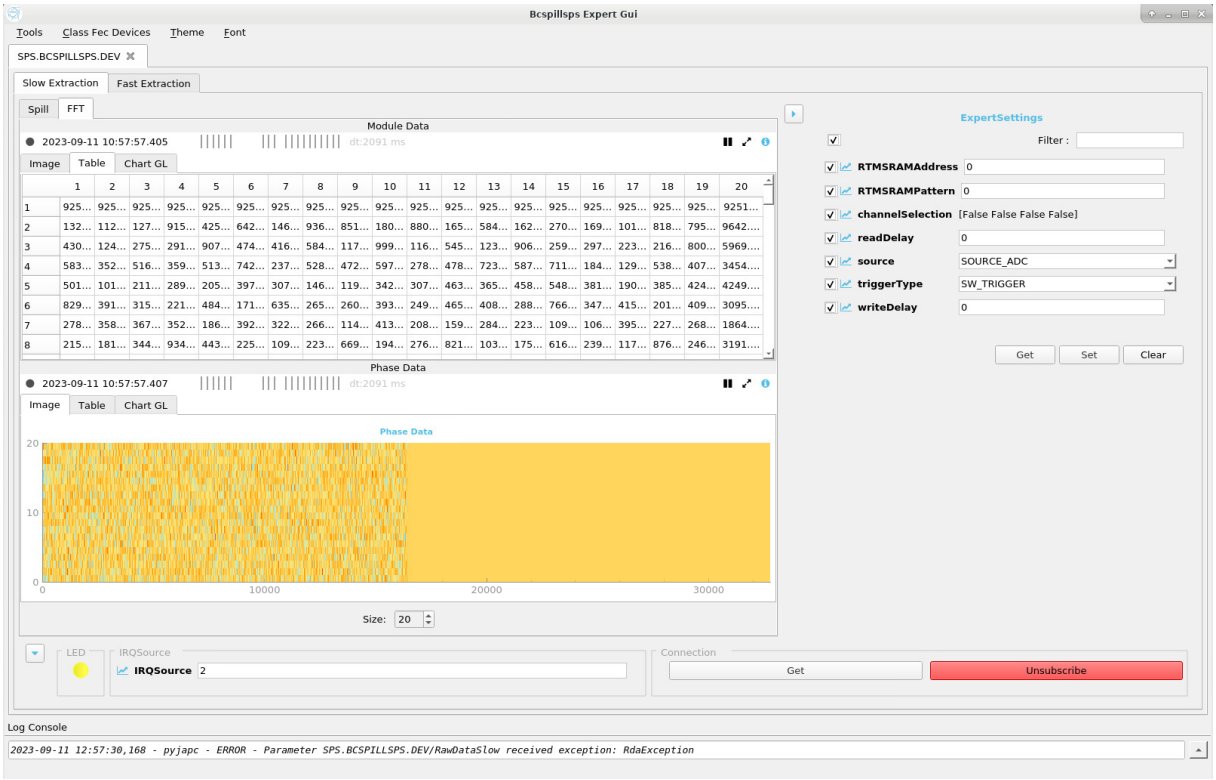

Figure 3: Current view of the BCSPILL SPS GUI

3.2 Software Architecture Overview

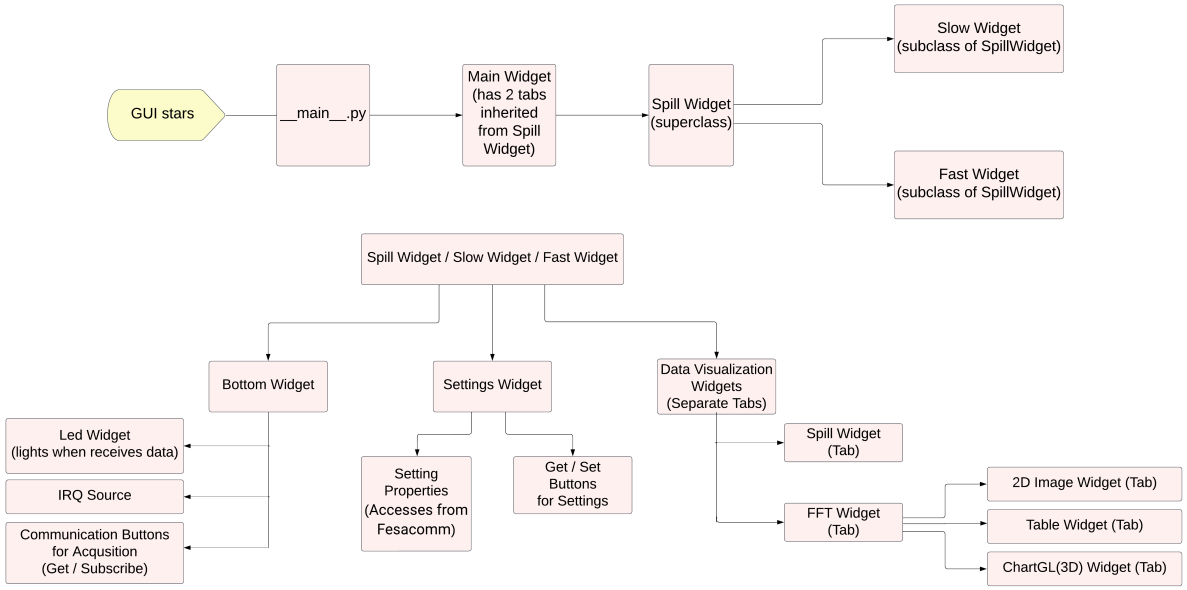

Figure 4: Communication Diagram

To initiate the GUI, the first step involves running the `__main__.py` file. This file takes care of initializing fundamental UI widgets such as the Menu Bar and Log Console before invoking the Main Widget.

The log panel allows you to view current and past logs.

Figure 5: Log Panel

The Menu Bar consists of the following options:

Tools: This option controls the visibility of plugins such as Timing and Log.

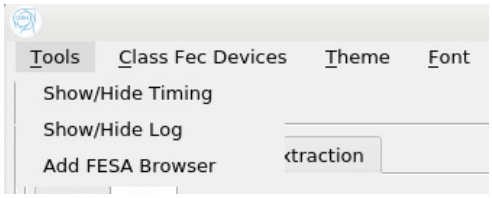

Figure 6: Tools

Class Front End Computer(FEC) Devices: This provides access to FEC Devices specifically developed for the BI Instrument.

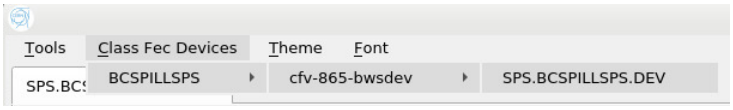

Figure 7: FEC Devices

Theme: This option allows users to switch between dark and light themes.

Figure 8: Theme

Font: Users can change the font settings through this option.

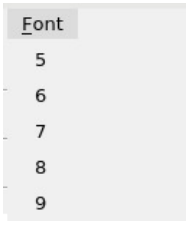

Figure 9: Font

The Main Widget is organized into two primary tabs: Slow Widget and Fast Widget. Both of these tabs inherit from the Spill widget, creating the core components of the user interface.

Figure 10: Tabs

Within the architecture, the Spill Widget serves as a pivotal component with three main parts:

1. **Visualization Widgets:**

   These widgets are responsible for rendering and displaying various forms of incoming data.

   There are two main visualization widgets:

   - **SPS Spill:** Visualizes incoming Acquisition data by creating plots.

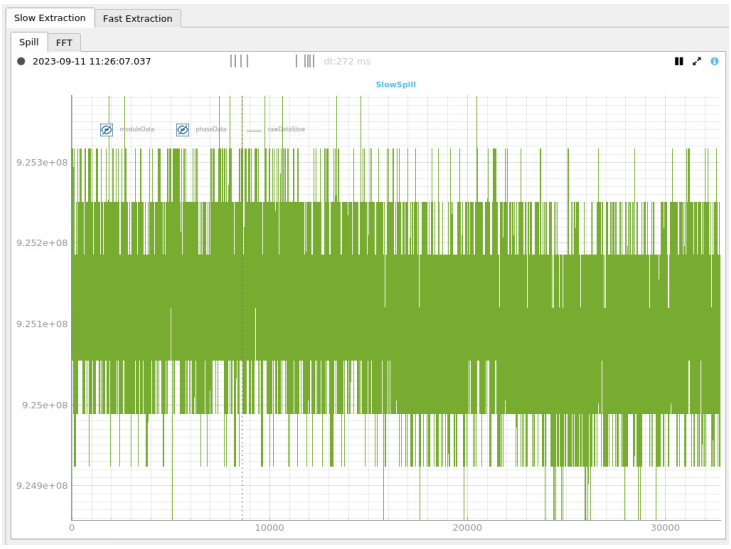

Figure 11: SPS Slow Extraction Spill

- **Fast Fourier Transformation(FFT):** Visualizes incoming FFT data by creating 2D images, 3D plots and tables.

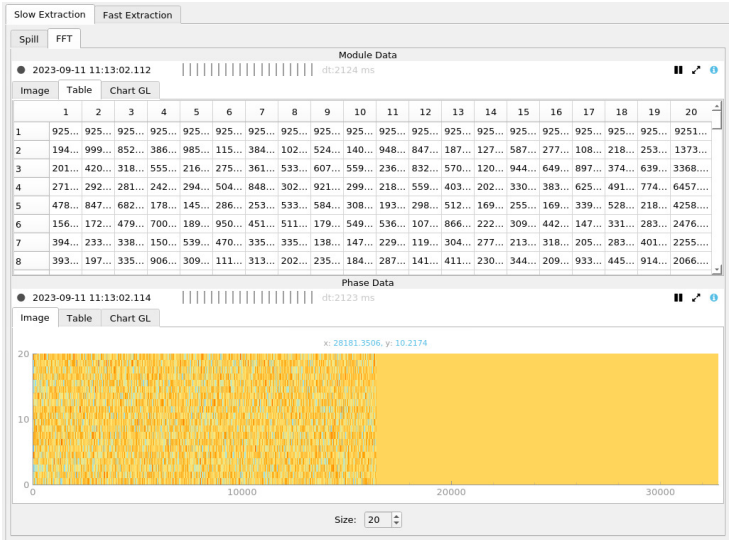

Figure 12: SPS Slow Extraction Spill FFT

2. **Settings Widget:**

   This widget establishes a connection with the FESA device's settings and enables the viewing and modification of these settings. The widget provides dedicated "Get" and "Set" buttons for interacting with these settings.

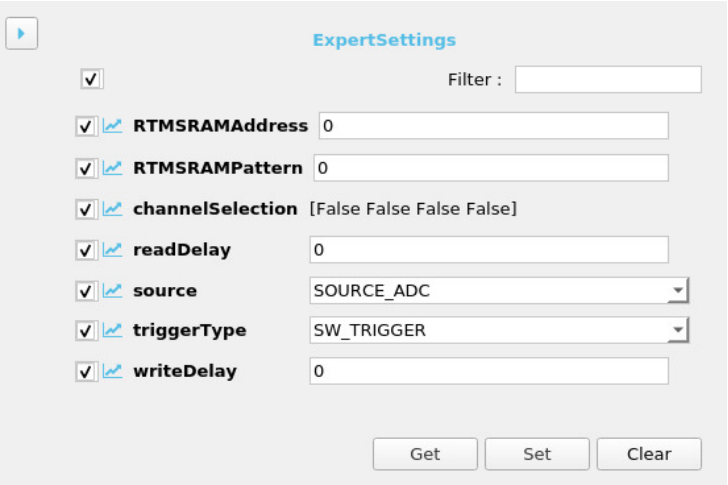

Figure 13: Settings

3. **Bottom Widget:**

   The Bottom Widget comprises three sub-widgets:

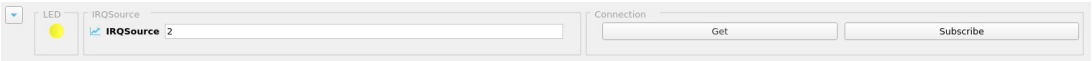

Figure 14: Bottom Widget

- **LED Widget:** Illuminates when data is received, serving as a visual indicator that the system and GUI are active.

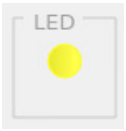

Figure 15: LED Widget

- **IRQ Source Widget:** Visualizes incoming Acquisition data by creating 2D images, 3D plots and tables.

Figure 16: IRQ Source Widget

- **Communication Buttons Widget:** Contains essential buttons, such as "Get" and "Subscribe," facilitating communication actions.

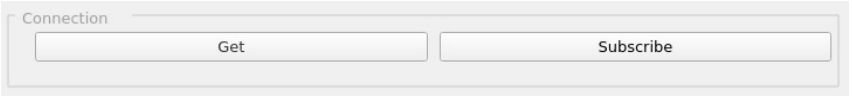

Figure 17: Buttons Widget

4 Software Implementation Concepts

- **Maintainability:**

  To ensure maintainability, I applied several best coding practices and followed a modular design approach. Breaking down the code into smaller, manageable modules allows for targeted updates and reduces the risk of unintended side effects when making changes.

- **Performance:**

  To achieve optimal performance, I used efficient libraries like NumPy. Leveraging NumPy's powerful array manipulation capabilities, data processing tasks are accelerated, leading to improved responsiveness and reduced computational overhead.

- **Standardization:**

  To achieve standardization, I carefully avoided reliance on non-standard libraries. By avoiding dependencies on external packages and libraries, the GUI reduced the risk of encountering compatibility issues with newer versions of those libraries.

- **Clean Code:**

  In adherence to the clean code principle, I followed the PEP8 style guide, which outlines coding conventions for Python projects. Additionally, comprehensive comments were added throughout the codebase to provide clarity on the purpose and functionality of different sections. This approach ensures that the code is easy to understand, and free from unnecessary complexity, and promotes maintainability.

Acknowledgements

I am deeply grateful to my supervisors Elif Balcı and Stephen Jackson, for their invaluable support and guidance throughout my summer student program at CERN. I'd like to express my gratitude to Stephane Bart Pedersen for his support on the expert-gui-core library and his valued feedback on my GUI. I would also like to extend my thanks to Ana Guerrero Ollacarizqueta, Steen Jensen, Mathieu Rodrigues Da Conceicao, Manuel Gonzalez Berges, Eirini Poimenidou, and all the members of our section. Being a part of this section has been truly an enriching experience.

Bibliography

[1] SY-DEP-BI. (n.d.). Software Section of the Beam Instrumentation Group. Retrieved from `https://sy-dep-bi.web.cern.ch/sw`

[2] PyQt. (n.d.). In Wikipedia. Retrieved from `https://en.wikipedia.org/wiki/PyQt`

[3] NumPy. (n.d.). NumPy Documentation. Retrieved from `https://numpy.org/doc/stable/`

[4] ACCSoft. (n.d.). ACCSoft GUI PyQt Widgets Documentation. Retrieved from `https://cern.ch/Widgets/wiki/`