# Software Framework for Real-time Flash Control and Data Visualization using NIDAQ-interfaced Power Supply

EMMANUEL BAMIDELE

---
COPYRIGHT INFORMATION
---


For questions, inquiries, and permissions related to the use and distribution of this software, please contact:

Emmanuel Bamidele
Email: correspondence.bamidele@gmail.com
Website: www.emmanuelbamidele.com
---
LICENSE INFORMATION
---


---
VERSION INFORMATION
---
Real-time Flash Experiment Control and Visualization software
Version: 1.0
Date: August 2023
---
DISCLAIMER
---
The Real-time Flash Experiment Control and Visualization software is provided "as is" without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and non-infringement. In no event shall the authors or copyright holders be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the software or the use or other dealings in the software.
The software is intended for research and educational purposes only. It is the responsibility of the user to ensure proper operation and adherence to all applicable laws and regulations while using this software.
---
CITATION INFORMATION
---
To cite the Software Framework for Real-time Flash Control and Data Visualization, please use the following citation:
Bamidele, Emmanuel. (2023). Software Framework for Real-time Flash Control and Data Visualization [Software framework]. Retrieved from https://github.com/Emmanuel-Bamidele/Real-Time-Flash-Control

For the GitHub repository, visit:
https://github.com/Emmanuel-Bamidele/Real-Time-Flash-Control

Documentation is available on ResearchGate.

# SUMMARY

Welcome to the Comprehensive Documentation for the Real-Time Flash Control Software! This documentation serves as your comprehensive guide to understanding, utilizing, and maximizing the potential of the Real-Time Flash Control software. Whether you're an experienced researcher seeking to optimize your flash experiments or a newcomer eager to grasp the nuances of the software, this documentation is tailored to cater to your needs.

**Software at a Glance** The Real-Time Flash Control software is a cutting-edge tool designed to enhance and streamline flash experiments. It offers real-time control, data acquisition, live plotting, and data storage capabilities, all wrapped within an intuitive graphical user interface (GUI). By delving into this documentation, you're embarking on a journey to harness the power of this software for advancing your research in materials science, physics, and engineering.

**Your Path to Mastery** As you navigate through this documentation, you'll explore every facet of the software, from its fundamental principles to its intricate functionalities. Each section is meticulously crafted to provide you with a detailed understanding of the software's architecture, features, and practical applications. Whether you're aiming to conduct flash experiments with precision or seeking to unravel transient phenomena through real-time data visualization, this documentation will equip you with the knowledge and tools to succeed.

**Unveiling Each Layer** From the user-friendly GUI to the seamless integration with data acquisition hardware, from live plotting that offers dynamic insights to the meticulous organization of your experimental data, each layer of the software will be peeled back, revealing the mechanisms that empower your research pursuits. By the end of this documentation, you'll not only comprehend the software but also confidently wield it to propel your research endeavors forward.

Are you ready to embark on this journey of exploration and mastery? Let's dive into the intricate world of the Real-Time Flash Control software and unlock its full potential for your research!

# 1. Introduction

In this section, you would introduce the purpose of the documentation, the software, and its relevance. You'd provide an overview of what the software does and the benefits it offers to researchers and scientists. The introduction sets the stage for the rest of the documentation by highlighting the importance of the software and its potential impact on flash experiments.

### 1.1 Purpose

The purpose of this documentation is to guide you through every aspect of the Real-Time Flash Control software. Whether you're an experienced researcher looking to optimize your flash experiments or a newcomer eager to understand the capabilities of the software, this documentation is tailored to meet your needs.

### 1.2 Scope

In this documentation, we will cover everything from the fundamental principles of flash experiments to the intricate details of the software's user interface, data acquisition methods, real-time visualization, and data storage. By the end, you'll be equipped with the knowledge and skills to confidently utilize the Real-Time Flash Control software for your research endeavors.

# 2. Software Overview

The Real-Time Flash Control software is a cutting-edge tool designed to enhance and streamline flash experiments. It offers real-time control, data acquisition, live plotting, and data storage capabilities, all within an intuitive graphical user interface (GUI). Let's explore its core features in detail:

### 2.1 Real-Time Control

The software empowers researchers to conduct flash experiments with precise control over experiment parameters. It enables real-time adjustments of parameters such as temperature, current density, voltage, and more, allowing researchers to closely monitor and influence the experimental conditions as they unfold.

### 2.2 Data Acquisition

Accurate data acquisition is at the heart of any successful experiment. The software interfaces seamlessly with National Instruments' Data Acquisition (DAQ) hardware to collect data from

multiple channels, including temperature, current, and voltage. This real-time data collection ensures that you capture every nuance of your flash experiment.

### 2.3 Live Plotting

Visualizing data in real-time is key to understanding transient processes. The software includes advanced live plotting capabilities that dynamically display data as it's acquired. You can choose the parameters you want to visualize and monitor their behavior in real-time. This feature aids in making informed decisions during the experiment.

### 2.4 Data Storage

Experimental data is precious and needs proper organization. The software allows you to store your acquired data in a structured manner. You can choose the destination folder and file naming conventions for your data files. This ensures that your valuable experimental results are stored efficiently for future analysis.

### 2.5 User-Friendly GUI

Navigating the software is effortless thanks to its intuitive GUI. Experiment parameters can be set easily through input fields, dropdown menus, and buttons. The software's clean design ensures that researchers of all levels can quickly become familiar with its functionalities.

### 2.6 Adaptability

Every experiment is unique, and the software acknowledges that. It can be tailored to suit a variety of experimental setups and requirements. Whether you're studying material properties, electrical phenomena, or other dynamic processes, the software can be configured to meet your specific needs.

Stay tuned as we delve deeper into each feature of the software, providing you with a comprehensive understanding of how to effectively use the Real-Time Flash Control software for your research goals.

## 3. Understanding the Application Components: Classes and Functions

In this section, we'll delve into the details of the key classes and functions that constitute the real-time data acquisition and monitoring application. Understanding these components is crucial for effectively utilizing, modifying, and extending the application to suit your research needs.

### 3.1 Classes Overview

#### 3.1.1 **AppRunModule** Class

The **AppRunModule** class encapsulates the primary functions for controlling the execution of the application. It consists of the following methods:

- **try_to_start()**: This method attempts to initialize the hardware and start the data acquisition process. It handles any exceptions that may arise during the initialization.
- **run_all()**: This method orchestrates the entire data acquisition process. It invokes functions to initialize hardware, perform the flash experiment, and finalize the acquisition. If any exceptions occur, they are caught and handled appropriately.
- **stop_running()**: This method halts the data acquisition process. It ensures that the hardware is properly stopped and releases any resources used.

#### 3.1.2 **PlotManager** Class

The **PlotManager** class is responsible for managing real-time data visualization on the live plot. It facilitates the creation and updating of plots with acquired data. Key methods and attributes include:

- **initialize_plots()**: Initializes the live plot by creating empty line plots for the primary and secondary axes.
- **update_plot_data()**: Updates the data of the existing line plots with the latest acquired data points and adjusts the plot limits for accurate visualization.
- **to_plot()**: Configures the axes and labels of the live plot based on user-selected parameters.
- **plot_strings(selected)**: Maps user-selected plot options to their corresponding data lists and axis labels.

### 3.2 Functions Overview

#### 3.2.1 Initialization and Control Functions

These functions are responsible for initializing hardware, updating control parameters, and executing the flash experiment.

- **initialize()**: Sets up hardware communication and establishes initial control parameters based on user inputs.
- **update_control()**: Updates control parameters based on user inputs, ensuring that values are accurate and within specified limits.
- **control_update_thread()**: Executes the **update_control()** function in a separate thread, allowing the GUI to remain responsive.

#### 3.2.2 Data Acquisition and Manipulation Functions

These functions handle data acquisition, manipulation, and communication with hardware instruments.

- **write_current(time_now)**: Writes the current value to the current channel on the DAQ hardware based on the current rate and time elapsed.
- **reset_current()**: Resets the current value to zero on the DAQ hardware.
- **read_voltage()**: Reads the voltage value from the voltage channel using the DAQ hardware.
- **read_temperature()**: Reads the temperature value from the temperature channel using the DAQ hardware.
- **flash()**: Executes the flash experiment by continuously collecting data, updating the live plot, and saving acquired data to a CSV file.
- **write_data()**: Creates and writes the data header to a CSV file where acquired data will be stored.

#### 3.2.3 User Interface Interaction Functions

These functions handle interactions with the graphical user interface (GUI) elements.

- **save_to()**: Opens a file dialog to select the folder where acquired data will be stored and updates the GUI accordingly.
- **screen_output(time_in, voltage1, resistance1, current_in, current_dense, temp)**: Updates the GUI front end output elements with real-time data values during the experiment.

### 3.3 Code Modularity and Interactions

The application's code is modular, allowing you to understand and modify specific components without affecting the entire application. The GUI part provides the user interface for interaction, while the non-GUI part contains the core logic for data acquisition, hardware control, and visualization.

Understanding the interactions and dependencies between classes and functions will enable you to confidently navigate, modify, and extend the application to meet your specific experimental requirements.

## 4. Detailed Explanation of Classes and Functions

### 4.1 The AppRunModule Class

#### 4.1.1 **try_to_start()**

This method attempts to initialize the hardware and start the data acquisition process. If any exceptions occur during hardware initialization, they are caught by the **try** block, and the **display_error_messages(ex)** function is invoked to show an error message to the user.

#### 4.1.2 **run_all()**

The **run_all()** method orchestrates the entire data acquisition process. It sequentially calls the necessary functions: **try_to_start()** to initialize hardware, **reset_current()** to set the current value to zero, **flash()** to execute the flash experiment, and **reset_current()** to reset the current value again. If any exceptions occur during any of these steps, they are caught and handled by invoking the **display_error_messages(ex)** function, and a message is displayed using **tkinter.messagebox.showinfo()**.

#### 4.1.3 **stop_running()**

The **stop_running()** method is responsible for halting the data acquisition process. It calls the **reset_current()** function to reset the current value to zero, closes the data file using **f.close()**, and exits the application using the **exit()** function. If any exceptions occur during these steps, they are caught and handled by invoking the **display_error_messages(ex)** function, and the application window is destroyed using **window.destroy()**.

### 4.2 The PlotManager Class

#### 4.2.1 **initialize_plots()**

This method initializes the live plot by creating two empty line plots using **ax1.plot()** and **ax2.plot()** for the primary and secondary axes, respectively. The resulting **line1** and **line2** objects will be used to update the plots with acquired data.

#### 4.2.2 **update_plot_data()**

The **update_plot_data()** method updates the data of the existing line plots (**line1** and **line2**) with the latest acquired data points. It also adjusts the limits of the axes using **ax1.relim()** and

**ax1.autoscale_view()** for accurate visualization. The method concludes by refreshing the plot using **plt.draw()**.

#### 4.2.3 **to_plot()**

The **to_plot()** method configures the axes and labels of the live plot based on the user-selected parameters for the x-axis, primary y-axis, and secondary y-axis. It extracts the selected options using **x_axis1.get()**, **pry_axis1.get()**, and **sec_axis1.get()**, and then maps these options to the corresponding data lists and labels using the **plot_strings(selected)** function.

#### 4.2.4 **plot_strings(selected)**

This function is a mapping between user-selected plot options and their corresponding data lists and axis labels. It returns the appropriate data list and label based on the selected option.

### 4.3 Initialization and Control Functions

#### 4.3.1 **initialize()**

The **initialize()** function sets up hardware communication and establishes initial control parameters based on user inputs. It initializes variables for channels, pyrometer temperature, voltage scaling, and more. It also calculates the current and voltage scales based on maximum current and voltage settings.

#### 4.3.2 **update_control()**

The **update_control()** function updates control parameters based on user inputs. It calculates the current limit based on the provided current density and area. If the calculated current limit exceeds the maximum allowable current, an error message is displayed in the GUI. This function is invoked in a separate thread using **control_update_thread()** to keep the GUI responsive.

#### 4.3.3 **control_update_thread()**

This function starts a separate thread to execute the **update_control()** function. This ensures that control parameter updates don't freeze the GUI and provides a smoother user experience.

### 4.4 Data Acquisition and Manipulation Functions

#### 4.4.1 **execution_time()**

The **execution_time()** function calculates and returns the total time required for the entire flash experiment based on the current limit and current rate.

#### 4.4.2 **write_current(time_now)**

This function writes the current value to the current channel on the DAQ hardware. It calculates the current value based on the current rate and the time elapsed since the start of the experiment. If the calculated current exceeds the current limit, the current limit value is used instead. The scaled current value is written to the DAQ hardware.

#### 4.4.3 **reset_current()**

The **reset_current()** function resets the current value on the DAQ hardware back to zero. This is important to ensure that the experiment starts with the correct initial conditions.

#### 4.4.4 **read_voltage()**

This function reads the voltage value from the voltage channel using the DAQ hardware. The raw voltage value is scaled based on the voltage scale calculated during the initialization.

#### 4.4.5 **read_temperature()**

The **read_temperature()** function reads the temperature value from the temperature channel using the DAQ hardware. The raw temperature value is read and returned.

#### 4.4.6 **flash()**

The **flash()** function executes the flash experiment. It continuously collects data by writing current, reading voltage and temperature, and updating the live plot. Data points are also saved to a CSV file. The live plot is updated with new data points, and the plot is refreshed to show the latest changes.

#### 4.4.7 **write_data()**

This function creates a CSV file to save acquired data. It writes the data header with column names and units to the file.

#### 4.4.8 **screen_output(time_in, voltage1, resistance1, current_in, current_dense, temp)**

The **screen_output()** function updates the GUI elements with real-time data values during the experiment. It populates the GUI fields with the latest values of time, voltage, resistance, current, current density, and temperature.

### 4.5 User Interface Interaction Functions

#### 4.5.1 **save_to()**

This function opens a file dialog to allow the user to select a folder where the acquired data will be stored. The selected folder's name is displayed on the GUI button.

#### 4.5.2 **folder_selected()**

When a folder is selected using the **save_to()** function, this function extracts the folder name from the selected path and updates the GUI button's text to indicate the current selected folder.

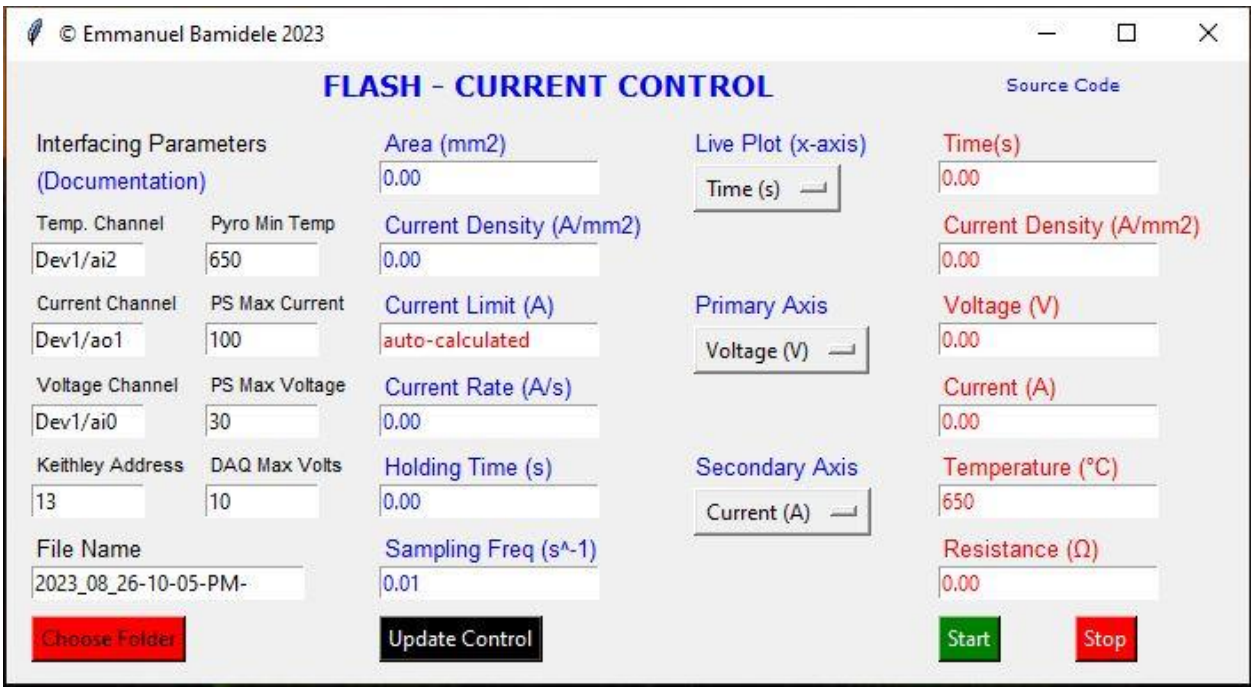
### 4.6 Code Modularity and Interactions

The interaction between the classes and functions creates a cohesive and functional application. The GUI interacts with the user, updating control parameters, and providing a visualization of the real-time data acquisition process. The non-GUI part handles the low-level interactions with hardware, data acquisition, manipulation, and storage.

Understanding the role of each class and function, along with their interactions, is essential for effectively utilizing and potentially modifying the application to suit specific research or experimental requirements.

## 5. Using the Software: Step by Step Guide

This section provides a detailed step-by-step guide on how to use the Real-Time Flash Control software for conducting flash experiments and acquiring real-time data.




### 5.1 Software Installation and Setup

1. **Download and Install Required Software:**
    - Make sure you have Python installed on your system. If not, download and install the latest version of Python from the official Python website.
    - Install the required Python packages by running the following command in your command-line interface or terminal:

```
pip install matplotlib nidaqmx pyvisa
```

2. **Download the Software:**
    - Download the Real-Time Flash Control software from the provided GitHub repository: https://github.com/Emmanuel-Bamidele/Real-Time-Flash-Control
    - Extract the downloaded ZIP file to a convenient location on your computer.
    - You can choose to install it as an app on your desktop or run from the python file.

3. **Hardware Setup:**
    - Connect the necessary hardware components, including the DAQ device, power supply, and any sensors or instruments required for your specific experimental setup.
    - Make sure the hardware is correctly connected and powered on.

### 5.2 Running the Software

1. **If already installed on your desktop as an application:**
    - Right click on the software and open.
2. **Run the Software using the python script:**
    - In the command-line interface, run the following command to start the Real-Time Flash Control software:

```
python flash_control_v1.py
```

    - The GUI window of the software will open, providing you with a user-friendly interface to control the flash experiment.

### 5.3 Configuring Experiment Parameters

1. **Interfacing Parameters:**
    - In the GUI, provide the required hardware interfacing parameters:
        - Temperature Channel: Enter the channel name for temperature measurement using the DAQ device.
        - Pyro Min Temp: Enter the minimum pyrometer temperature.
        - Current Channel: Enter the channel name for current control using the DAQ device.
        - PS Max Current: Enter the maximum power supply current.
        - Voltage Channel: Enter the channel name for voltage measurement using the DAQ device.
        - PS Max Voltage: Enter the maximum power supply voltage.

- Keithley Address: Enter the address of the Keithley instrument.

2. **Flash Control Parameters:**
    - Enter the area of the sample (in $mm^2$) and the desired current density (in $A/mm^2$) you wish to reach.
    - Specify the current rate (in A/s) at which the current should change during the flash experiment.
    - Set the holding time (in seconds) for which the current is held constant.
    - Define the sampling frequency (in Hz) for data acquisition.
    - Do not fill anything in the current limit, it will be auto calculated. If the value calculated exceed the limit of the power supply, "Limit Exceeded" will be returned. You can adjust value and re-update.
3. **File Naming and Storage:**
    - Enter a desired file name for the data file that will store acquired data. The default name includes the current date and time.
    - Click the "Choose Folder" button to select a folder where the data file will be saved.
    - The selected folder will be displayed after selection. You can change it before acquiring data.
4. **Live Plot Configuration:**
    - Configure the x-axis, primary y-axis, and secondary y-axis for the live data plot.
5. **Update Control Parameters:**
    - Click the "Update Control" button to apply the specified experiment parameters and update the current limit.

### 5.4 Running the Flash Experiment

1. **Starting the Experiment:**

- After filling all parameters and ensuring you have updated the control, click the "Start" button to begin the flash experiment. The software will start acquiring data and updating the live plot.

2. **Monitoring the Experiment:**
    - As the experiment progresses, real-time data, including time, voltage, current, resistance, and more, will be displayed on the GUI.
3. **Live Plot Updates:**
    - The live plot will continuously update with new data points, providing a visual representation of the experiment's progress.

### 5.5 Finishing the Experiment

1. **Stopping the Experiment:**
    - Click the "Stop" button to halt the flash experiment. The software will reset the current value and close the data file.
2. **Data Storage:**
    - The acquired data is saved in a CSV file in the specified folder. You can access this file to analyze and process the data further.

### 5.6 Handling Errors

1. **Error Messages:**
    - If any errors or exceptions occur during hardware initialization, data acquisition, or any other step of the process, the software will display appropriate error messages. Read and understand the error messages to diagnose and address the issue.
2. **Exiting the Software:**
    - You can close the GUI window by clicking the close button (X) in the corner of the window.

### 5.7 Software Modifications and Customization

The Real-Time Flash Control software is designed to be customizable to suit various experimental setups. Advanced users familiar with Python programming can modify and extend the code to include additional features or adapt it for specific requirements. Before making any modifications, ensure you have a good understanding of the code and its functionality.

## 6. Conclusion and Future Enhancements

In this comprehensive documentation, we have explored the Real-Time Flash Control software, its underlying structure, and its functionality. The software empowers researchers and scientists to conduct flash experiments with real-time data acquisition and visualization, aiding in the analysis and understanding of transient processes.

By providing a user-friendly GUI, efficient data acquisition, live plotting, and data storage capabilities, this software streamlines the flash experiment workflow and enhances the accuracy of experimental results. It offers a flexible platform that can be adapted to various experimental setups and research domains.

### 6.1 Future Enhancements

As technology and research needs evolve, there are several areas where the Real-Time Flash Control software can be enhanced and extended:

1. **Advanced Data Analysis:** Incorporating advanced data analysis and visualization tools directly within the software, allowing users to perform preliminary analysis and gain insights during and after the experiment.
2. **Experiment Templates:** Providing pre-defined experiment templates for common flash experiments, enabling users to quickly set up experiments without the need to manually configure all parameters.
3. **Multi-Platform Support:** Adapting the software for cross-platform compatibility, enabling its use on different operating systems without the need for significant modifications.
4. **Integration with Online Community Support:** One exciting prospect is the integration of the software with an online community forum. While this feature is currently under consideration, the idea is to create a space where users can connect, share experiences, and exchange insights. This community-driven platform could serve as a valuable resource for troubleshooting, idea-sharing, and fostering a collaborative environment among users.

5. **Modular Architecture:** Refining the software's architecture to enhance its modularity, making it easier to add, remove, or modify specific features.
6. **Real-Time Analysis:** Integrating real-time analysis capabilities that allow users to perform basic data processing and visualization during the experiment, facilitating quick decision-making.
7. **Enhanced Error Handling:** Further improving error handling mechanisms to provide more detailed and context-specific error messages, aiding users in diagnosing and resolving issues.
8. **Community Contribution:** Encouraging contributions from the research community to enhance the software's capabilities, add new features, and address specific research needs.

### 6.2 Conclusion

The Real-Time Flash Control software offers a powerful and flexible platform for conducting flash experiments and acquiring real-time data. With its intuitive GUI, efficient data acquisition, and live plotting features, researchers can gain deeper insights into transient processes and phenomena. As research in materials science, physics, and engineering continues to progress, this software stands as a valuable tool to advance our understanding of dynamic events.

We hope this documentation has provided you with a comprehensive understanding of the software's features, functionality, and potential applications. Whether you are a seasoned researcher or a newcomer to flash experiments, we encourage you to explore, experiment, and contribute to the ongoing development of this valuable research tool.

Thank you for choosing the Real-Time Flash Control software for your research endeavors.