Mass Spectrometer Interface

A Desktop Application for Reading Instrument Data

Cousins Photosynthesis Lab in the School of Biological Sciences at WSU



**Team Linnaea Borealis**

****

Kyler Kupp, Erik Holtrop

**TABLE OF CONTENTS**

[I. Introduction 3](#_Toc82521031)

[II. System Overview 3](#_Toc82521032)

[III. Architecture Design 3](#_Toc82521033)

[III.1. Overview 3](#_Toc82521034)

[III.2. Subsystem Decomposition 3](#_Toc82521035)

[I.1.1. [Subsystem Name] 4](#_Toc82521036)

[a) Description 4](#_Toc82521037)

[b) Concepts and Algorithms Generated 4](#_Toc82521038)

[c) Interface Description 4](#_Toc82521039)

[I.1.2. [Include sections III.2, III.3, etc., for other subsystems] 4](#_Toc82521040)

[IV. Data design 4](#_Toc82521041)

[V. User Interface Design 4](#_Toc82521042)

[VI. Glossary 5](#_Toc82521043)

[VII. References 5](#_Toc82521044)

[VIII. Appendices 5](#_Toc82521045)

# Introduction

This document aims to outline our project’s approach. This includes the different parts of the project modules (Architecture Design), data storage and manipulation methods (Data Design), and how the project modules will outwardly look (User Interface Design). These details are intended to elucidate objectives and methods for current and future developers, but to do so using language that can mostly still be understood by stakeholders for overview. Our project is split into a few modules. The first 3 modules offer different perspectives on Mass Spectrometer data through graphs and logs, the 4th one reformats data from a 2nd Mass Spectrometer, and the 5th one combines data from three different instruments’ data-streams. We focus on the 5th module when it comes to architecture design, as that’s where we have to do the most original design.

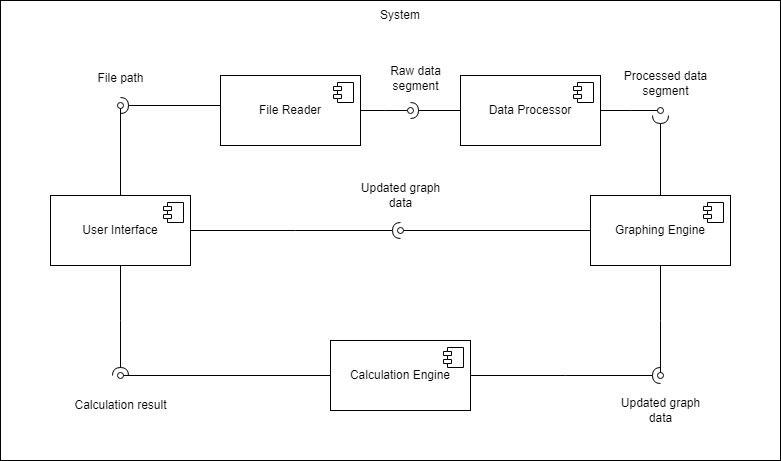
# System Overview

The project is designed to provide a comprehensive platform for managing and interpreting Mass Spectrometer data through various modules. It aims to present data insights through graphical and numerical perspectives while offering flexible data formatting and integration capabilities. The system consists of five primary modules: the first three focus on different ways to visualize and log Mass Spectrometer data, the fourth module reformats data from a secondary Mass Spectrometer, and the fifth combines data streams from three distinct instruments. The general design for each of the modules is to accept input from a file or folder, which is selected by the user, process it in some way (reformat, normalize, derive calculations from, etc.), and then output that processed information, either to graphs on the screen, or to new files.

# Architecture Design

## Overview

Team Linnaeus' architecture design focuses on Module 5. Each module in the system operates as a standalone program, but Module 5 is the primary module requiring architectural design innovation, as most other modules remain stable and do not necessitate changes. The team has adopted a Pipe and Filter architecture for this design, which is particularly well-suited for this system's data processing requirements. The system requires data to be read from a file, normalized, plotted, and then transformed. This process fits Pipe and Filter architecture well. This approach allows for clear separation of concerns, where data flows through a series of independent filters (or stages), each performing specific tasks such as reading, processing, and visualizing data. This modular design enhances flexibility, scalability, and maintainability, making it easier to integrate new features or modify existing ones without affecting the entire system. By leveraging this architecture, Team Linnaeus ensures a clean and efficient flow of data from input to output, supporting both current and future needs. The system begins with the User Interface (UI) component, which allows users to specify a file path and view plotted data and calculation results. The UI interacts with the File Reader, which reads raw data from the CSV file provided by the user and passes it to the Data Processor. The Data Processor then normalizes the raw data by applying a natural logarithmic transformation, preparing it for further steps. Once processed, the data is sent to the Graphing Engine, which updates the current graph by integrating the new data. The updated graph is then displayed in the UI for the user and sent to the Calculation Engine. The Calculation Engine performs further computations on the processed data, producing analytical results that are also displayed in the UI such as the mean of a selection of data. This flow of data, facilitated by the modular design, ensures efficiency, flexibility, and ease of maintenance.

Figure 1: System Block Diagram

## Subsystem Decomposition

This section outlines how the system has been decomposed into its major subsystems, each corresponding to the core components identified earlier: the User Interface (UI), File Reader, Data Processor, Graphing Engine, and Calculation Engine. The decomposition was designed to ensure that each subsystem represents a manageable unit of work for a single developer, with clearly defined responsibilities and minimal overlap. Each subsystem has been assigned a specific functionality: the UI handles user interactions, the File Reader manages data input, the Data Processor handles transformations, the Graphing Engine visualizes the data, and the Calculation Engine performs analytical operations. The rationale behind this decomposition emphasizes cohesion and coupling. High cohesion is maintained within each subsystem by ensuring that each is focused on a single task or closely related set of tasks. Meanwhile, coupling between subsystems is minimized by defining clear and straightforward interfaces for data exchange, allowing for seamless interaction without unnecessary dependencies. This modular design not only improves the maintainability and scalability of the system but also supports flexible integration of future enhancements.

### User Interface

#### Description

The User Interface (UI) subsystem is responsible for managing the interaction between the user and the system. Its primary function is to display graph data generated by the graphing engine as the results of calculations performed by the Calculation Engine. Additionally, the UI allows the user to input the file path of the CSV data, which is then passed to the File Reader subsystem for processing. The UI acts as a central hub, presenting outputs from different subsystems in a coherent and user-friendly manner.

#### Concepts and Algorithms Generated

The User Interface subsystem was designed with simplicity and ease of use in mind, utilizing a basic file input mechanism and graphical display capabilities. Concepts such as event-driven programming were considered, allowing the UI to react to user inputs (e.g., file path selection) and update dynamically when data from other subsystems (such as the graph or calculations) is received. The selected solution leverages standard UI frameworks to achieve these goals with minimal latency and high responsiveness. The decision to keep the UI simple was influenced by the need to maintain a clear separation of concerns, ensuring that complex data processing is handled by other subsystems. Trade-offs included balancing performance with usability, ensuring the UI remains responsive even when handling large datasets.

#### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| GetFilePath | File Reader | The GetFilePath service will pass over the user-selected file paths for data acquisition. Three file paths should be returned, one for each instrument utilized by the system. |
| GetDataSelection | Calculation Engine | This service will return the bounds of the data selected by the user using two movable vertical bars. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| CalculateMean | Calculation Engine |
| GetLatestGraph | GraphingEngine |

### File Reader

#### Description

The File Reader subsystem is responsible for reading data from three different files, each representing data from a distinct instrument used by the system. After reading the raw data from these files, the File Reader passes the data along to the Data Processor for further processing. This subsystem acts as the initial stage in the data pipeline, ensuring that the raw information is properly retrieved and made available for downstream components.

#### Concepts and Algorithms Generated

The File Reader subsystem was designed with the need to efficiently handle multiple file inputs. The concept of parallel file reading was considered to improve performance when handling large datasets, but ultimately, a sequential approach was selected due to the relatively manageable file sizes. The File Reader ensures that each file is opened, its contents are read, and the data is collected into a format that can be easily passed to the Data Processor. Special considerations were made to handle potential file reading errors, such as missing or corrupted files. In such cases, error handling mechanisms trigger notifications to the user. The trade-off involved in this design was balancing robustness with simplicity, ensuring that file reading remains a lightweight process while effectively managing edge cases.

#### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| ReadFileData | Data Processor | This service reads the contents of the three files, each corresponding to an instrument's data. The service accepts the file paths as input (provided by the User Interface) and returns the raw data from all three files as output. The data is passed on to the Data Processor for formatting and normalization. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetFilePath | User Interface |

### Data Processor

#### Description

The Data Processor subsystem is responsible for transforming raw data received from the File Reader into a format that is more suitable for further analysis. Its primary task is to normalize the data using the natural logarithmic transformation, making the data easier to process and visualize. Once the data is formatted and normalized, the Data Processor sends the processed data segment to the Graphing Engine, where the new data is added to the current graph. This subsystem ensures the integrity and usability of the data as it flows through the system.

#### Concepts and Algorithms Generated

The main algorithm used by the Data Processor is the natural logarithmic transformation, which helps normalize the data. This technique was selected because it reduces variance in the data and handles large discrepancies in value magnitudes, making it more suitable for visual representation and computation. Alternative normalization methods, such as z-score normalization, were considered but ultimately not chosen, as the natural log method better aligned with the requirements of the data's structure and scale. Special considerations included handling cases where the raw data contains zero or negative values, as these would cause issues during the logarithmic transformation. Error handling mechanisms and data validation were incorporated to account for such anomalies, ensuring robust processing. The trade-off in this approach involved balancing data transformation speed and the complexity of handling edge cases.

#### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| ProcessData | Graphing Engine | This service receives the raw data from the File Reader, processes it by formatting and applying a natural logarithmic normalization, and then sends the processed data to the Graphing Engine for visualization. The input consists of the raw data from three different files, and the output is the normalized and formatted data ready for graphing. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| ReadFileData | File Reader |

### Graphing Engine

#### Description

The Graphing Engine subsystem is responsible for visualizing the processed data received from the Data Processor. Its main role is to update the current graph by incorporating the newly processed data segment and rendering it in the user interface. This allows the user to observe real-time visual changes in the data as it is processed. The Graphing Engine ensures that the graphical representation is clear, accurate, and continuously updated as new data is received.

#### Concepts and Algorithms Generated

The Graphing Engine employs algorithms to dynamically update the graph with new data segments. Various graphing techniques were considered, including line charts and bar charts, but a continuous line chart was selected as the best fit for real-time data visualization. The primary concept is to ensure that the graph scales efficiently as more data is added, while also maintaining clarity and responsiveness.

#### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| UpdateGraph | User Interface | This service receives the processed data from the Data Processor and updates the current graph displayed in the User Interface. The input is the normalized data, and the output is an updated visual graph reflecting the latest data. The service ensures that the graph is refreshed in real-time as new data is processed. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetProcessedData | Data Processor |

### Calculation Engine

#### Description

The Calculation Engine subsystem is responsible for performing various calculations on the data, including allowing the user to select a segment of data and calculating the mean of that segment. This subsystem plays a crucial role in providing numerical insights into the processed data, complementing the visual representation provided by the Graphing Engine.

#### Concepts and Algorithms Generated

The primary algorithm used by the Calculation Engine is the calculation of statistical measures such as the mean. The mean is calculated by summing the selected data points and dividing by the number of points. Several approaches were considered, including pre-calculating statistics for all data and storing them for quick access. However, the selected solution was to compute the required statistics on-demand based on the user’s selection. This minimizes resource consumption and ensures that calculations are based on the latest data segment selected by the user.

#### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| CalculateMean | User Interface | This service calculates the mean of the selected data segment. The input is the processed data for the user-selected segment, and the output is the mean value. The mean is displayed in the User Interface as part of the calculation results. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetDataSelection | User Interface |
| GetLatestGraph | GraphingEngine |

# Data design

This section covers the different ways our project stores data, both while it’s running, and when the data is stored for later. This overview will be a fairly technical overview, meant mostly for developers. Each subsection covers a different data structure used in the project.

## IV.1 Shared Singleton

A shared singleton is a design pattern that forces only one instance of an object to exist throughout the whole program. Although this design pattern does not usually describe a data structure, in modules 1-3, most important data structures are members of a shared singleton, named sharedData. These members include fileList (subsection IV.2) and dataPoints (subsection IV.3)

## IV.2 Basic List

In modules 1-3, we use a basic list to store the names of files that are read from. This list is named fileList and is stored in the sharedData shared singleton (subsection IV.1).

## IV.3 Dictionary

In modules 1-3, we use a Python dictionary to store the mass spectrometer data, named dataPoints. The keys are time points, and the values are tuples of isotope masses. This dictionary is stored in the sharedData shared singleton (subsection IV.1).

## IV.4 Log Table

In modules 1-3, there are certain data points that can be pulled out from the graph, or otherwise calculated from the data. These can be saved into a table built into the UI. These logs are then stored directly in the PyQT UI component QTableWidget.

## IV.5 CSV Export

Modules 1-3 can export logged data from the log table (subsection IV.4) into a CSV file, which matches the format of the table.

## IV.6 Pandas DataFrame

Module 4 uses a DataFrame from the Pandas Python library to store data from Mass Spectrometer 2. This is very similar to a dictionary approach (subsection IV.3), but effectively replaces the need for unique keys with ordering and more importantly allows for the use of Pandas methods like to\_csv at a small performance cost.

## IV.7 CSV Series

Module 4 exports the DataFrames (subsection IV.6) as CSV’s with ~8 rows each, for use in modules 1-3. Each row contains a time signature and isotope masses. These CSV’s are named with numbers in order of their time signatures.

# User Interface Design

The user interface will first allow the user to select acquisition folders for data from the Mass Spectromer, for modules 1-4, or for data from each of the three instruments, in the case of module 5. It will share the same design as shown in Figure 2. However, it will have three separate options for each type of data that the system utilizes. This portion of the interface corresponds with use case UC-4: Select Input Files.

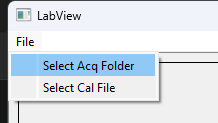
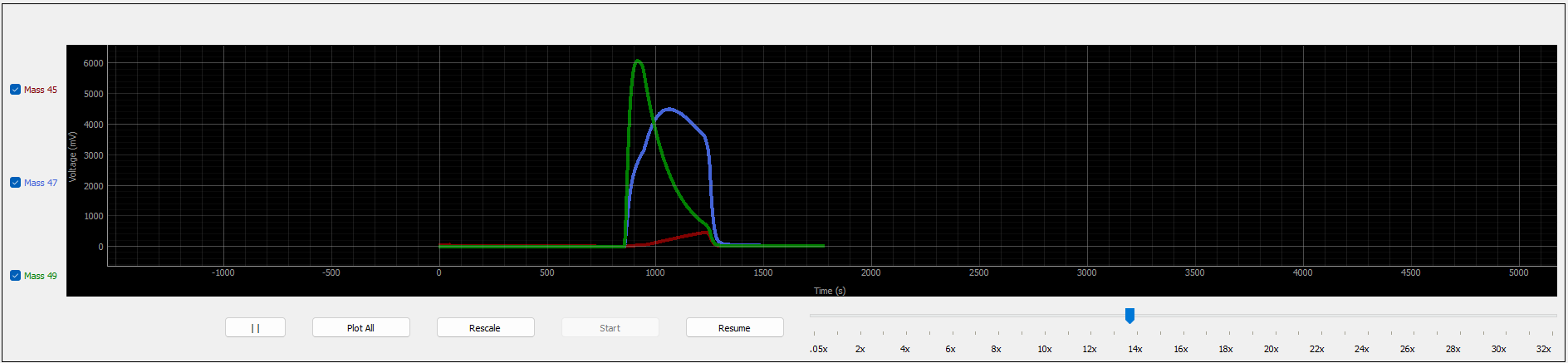
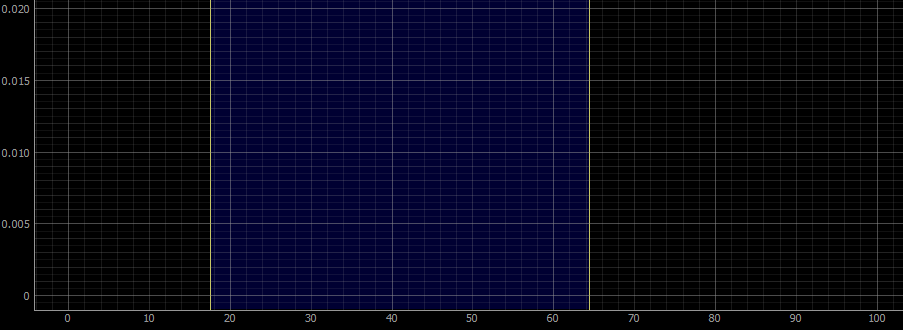


Figure 2: File Selection

The largest section of the user interface will display a black graph such as the one shown in Figure 3. To the left of the graph, color coded labels for each data stream are present to provide an intuitive viewing experience. The graph is populated by line-graphs, color coded for each data stream. This corresponds to the use case UC-5: View Graph. The graph will be easily scalable using the scroll wheel to zoom in or out. Below the graph and to the right of Figure 3 a slider is shown. This slider allows the user to adjust the speed at which data is plotted from 0.5x to 32x. This corresponds to use case UC-7: Change Plotting Speed. On the left of the slider is also a pause/resume button as well as a start button that allows the user to toggle the plotting of data. In addition to the primary graph, modules 1-3 have additional graphs for calculations derived from the primary data.

Figure 3: Graphing and Calculations

Using the leftmost button in Figure 3 that features two vertical lines, the user can toggle the visibility of “mean bars”. These vertical bars pictured in Figure 4 can be shifted left to right using the cursor to select a specific portion of data. This corresponds to the use case UC-8: Select Data Points.

Figure 4: Data Segment Selection

Available calculations will be featured below the graph and its controls. Here the user will be able to use the “Get Mean” button to calculate the mean inside of the selected region. This corresponds to use case UC-9: Calculate Mean.



Figure 5: Mean Calculation

In addition to allowing users to scale the graph using their scroll wheel, users will be able to achieve more precise scaling by left clicking the graph to open further adjustment options. As illustrated in Figure 6, users will be able to input the exact bounds of each axis, and the graph will adjust according to their selection.

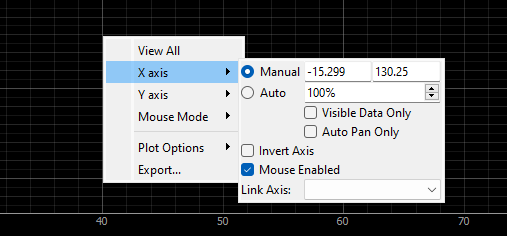


Figure 6: Detailed Graph Scaling

Overall, the user interface is designed for an intuitive data acquisition and visualization experience. Users can select acquisition folders for three instruments, with options tailored for each data type, as shown in Figure 2 (UC-4: Select Input Files). The main area features a black graph displaying color-coded line graphs, accompanied by labeled indicators for clarity (UC-5: View Graph). Users can zoom using the scroll wheel and adjust the plotting speed with a slider (UC-7: Change Plotting Speed). Controls for toggling mean bars and selecting data points further enhance functionality (UC-8: Select Data Points), while a “Get Mean” button allows for quick calculations of averages within selected regions (UC-9: Calculate Mean). Additionally, precise graph scaling options enable users to customize their view, ensuring a comprehensive and user-friendly interface.

# Glossary

Define technical terms used in the document.

# References

(Dutoit, 2010), 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice Hall, 2010.

Cite your references here.

For the papers you cite give the authors, the title of the article, the journal name, journal volume number, date of publication and inclusive page numbers. Giving only the URL for the journal is not appropriate. You should use either IEEE or Chicago style formatting for your citations

For the websites, give the title, author (if applicable) and the website URL.

# VIII. Appendices

Any larger images, charts, or external materials should be put into appendices. These are attached at the end of the document, so the main materials are kept closer together and the overall flow of the document is preserved. If you include 4 pages of spreadsheets in the middle of a section, it makes it very difficult to track the flow of your presentation. Instead, those sheets go in Appendix [X] and are referred to by the earlier document.

You may have as many appendices as you need for the document to make sense.