

SWATHLib  
Technical Manual and Common Usage

Toan K. Phung

November 20, 2018

# Contents

<b>Contents</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 General Automated Workflow Summary . . . . .	1
1.2 GUI System Requirement . . . . .	1
1.3 Back-end System Requirement . . . . .	2
<b>2 Graphical User Interface</b>	<b>3</b>
2.1 Back-end Connection . . . . .	3
2.2 User Input . . . . .	4
<b>3 Back-end</b>	<b>8</b>
3.1 User Query Preprocessing . . . . .	8
3.2 Theoretical Ion Generation . . . . .	8
<b>4 Common Usage Scenarios</b>	<b>9</b>

# Chapter 1

## Introduction

SWATHLib is an attempt at creating an automated workflow for creation of manually curated spectral library for SWATH downstream analysis. The program itself is composed of two main component, the front-end created using TypeScript with Electron and the Angular framework and the back-end created using Python with Tornado as the server implementation.

### 1.1 General Automated Workflow Summary

1. Input of user sequences and select parameters within the GUI (e.g SWATH windows, retention time (RT), modifications, result output format...).
2. Submission of user's queries to SWATHLib back-end.
3. Check whether there are any modifications and change the workflow settings accordingly.
4. The program attempt to generate possible combination of all the modifications from user query.
5. For each combination, transition sequences of  $Y$  and/or  $b$  and/or  $y$ -series would be created and their  $m/z$  values calculated.
6. Transition with  $m/z$  within **50-1800  $m/z$**  range would be recorded for each combination of RT and window.
7. If oxonium ions were included within the queries, for each unique combination of precursor, RT and window, entries for oxonium ions would be recorded for them that combination.
8. The end results would be communicated to the user for retrieval.

### 1.2 GUI System Requirement

#### Windows

- Windows 7 or later

- Intel Pentium 4 processor or later with SSE2
- 512 MB of RAM

### **Mac**

- OS X 10.9 and later
- 64-bit Intel processor
- 512 MB of RAM

### **Linux**

- Debian 8, Fedora 21, Ubuntu 12.04 and later
- Intel Pentium 4 processor or later with SSE2

## **1.3 Back-end System Requirement**

- Any system that can support Python 3.5+ with Tornado v5+ and Pyteomics v3.5.1+ packages.

## Chapter 2

# Graphical User Interface

The GUI at its core is a chromium web instant that load a single page JavaScript application. The GUI is only necessary to communicate user input and retrieve result from back-end. It is not needed if the user only want to run the back-end.

### 2.1 Back-end Connection

At the top right is a button that can be used to access the back-end connection management windows where the user can start an instant of the Python back-end and manually adding more server connections to split user's the workload.

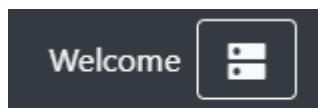


Figure 2.1: Button for access to the back-end connector management panel.

The connector management panel has two sections. The first section is for running an instance of the back-end on the current computer. The input requires the absolute file path to the python binary that have access to Tornado v5+ and Pyteomics v3.5.1+, and the port number that the program would be run on.

In the second section of the panel, the user can enter addresses to one or more SWATHLib back-end instance. The GUI would check every saved addresses for accessibility by performing a GET request to the address at the path `/api/swathlib/upload/`. If the response status is 200, the address is accessible else, it would be inaccessible and would not be used in the workflow.

## 2.2 User Input

The GUI accepts protein sequences in FASTA format either in raw text or a text file. The user can perform in-silico trypsin digestion of the input with optional ability to manually or automatically select sites for misdigestion.

### Modification Customization

Within user settings tab, the user can use their own table of modifications in tabular text file format as modifications source for the program. The table is made up of 10 columns where,

- **name** is the name of the modification to be displayed on the GUI.
- **label** is the internal label of the modification to be used by the back-end. Must be in lowercase. No modifications within the same query should have the same label beside transition of the same Ytype variable modification.
- **mass** is the mass of the modification.
- **regex** is the regular expression representation of the the modification.
- **type** is the modification type *Ytype*, *variable*, or *static*.
- **Ytype** is the label of the Ytype transition. Only applicable if type is Ytype.
- **multiplepattern** is whether the query with this variable modification is default to have all modification pattern generated. Only applicable if type is Ytype or variable.
- **status** is whether the query with this variable modification is default to be filled in every instance on the sequence. Only applicable if type is Ytype or variable.
- **mlabel** is the label for static modification in the annotated sequence.
- **offset** is the value offset for motif finding in case the digested fragments are cut within the motif. Only applicable with the GUI tryptic digestion tool.

Table 2.1: Example of a user submitted modifications table

name	label	mass	regex	type	Ytype	multiplepattern	status	mlabel	offset
O-Mannose	g	0	[S T]	Ytype	Y0	false	false		0
O-Mannose	g	162.05	[S T]	Ytype	Y1	false	false		0
HexNAc	h	0	N[ <sup>-</sup> P][S T]	Ytype	Y0	false	false		2
Propionamide	c	71.037114	C	static		false	false	PPa	0
Carboxylation	e	43.98983	E	variable		false	false		0

### SWATH Window Customization

Similar to modification customization, the same settings tab, the user can also replace the default SWATH windows with an array of customized range using a tabular text file with two column, the first is the starting m/z of the windows while the second column is the stopping m/z of the windows.

Table 2.2: Example of a user submitted SWATH window table

start	stop
400	425
424	450
449	475
474	500
499	525
524	550
549	575
574	600
599	625
624	650
649	675
674	700
699	725

### Retention Time Customization

Within the user settings tab, a customized retention time list could be entered into the text box, each RT separated by a newline. Upon saving, the customized time would replace the default time within the GUI. Currently we only support time in integer.

### Main Input Settings

Main input settings composed of 10 input parameters.

- **Static Modifications**, modifications that will always be on the precursor sequences.
- **Non Ytype Variable Modifications**, variable modification that is not Ytype and may or may not appear on its supposed modification sites on the precursor sequences.
- **Ytype Variable Modifications**, variable modification that is Ytype and may and may or may not appear on its supposed modification sites on the precursor sequences. Multiple Ytype transition of the same modification can be applied without raising modification conflict issues.
- **Retention Time**, retention time for the library to be generated at.
- **SWATH Windows**, SWATH windows for the library to be generated at.

- **Max Charge**, max transition charge for the library to be generated at.
- **Precursor Charge**, charge of the precursor, only used in calculation of precursor  $m/z$  if no SWATH windows were selected.
- **Oxonium Ions**, ion fragments of a Ytype modification.
- **Fragmentation ion-type**, can be a string combination of  $b$ ,  $y$ , and  $Y$ , dictate which transition series that the library would attempt to generate.
- **Output Sequence Format At Variable Modifications**, dictate how the precursor annotated sequences would be uniquely identified across different RT and SWATH windows combination from each other at positions of variable modifications or at the end of the sequence. The three option is RT or SWATH windows or RT and SWATH windows.

These settings would apply to all sequences selected by the user at the beginning upon clicking the *Apply Modifications* button and replaced any settings set before.

### Individual Query Settings

Beside the main settings, the user can also customize a few parameters for each individual query sequence.

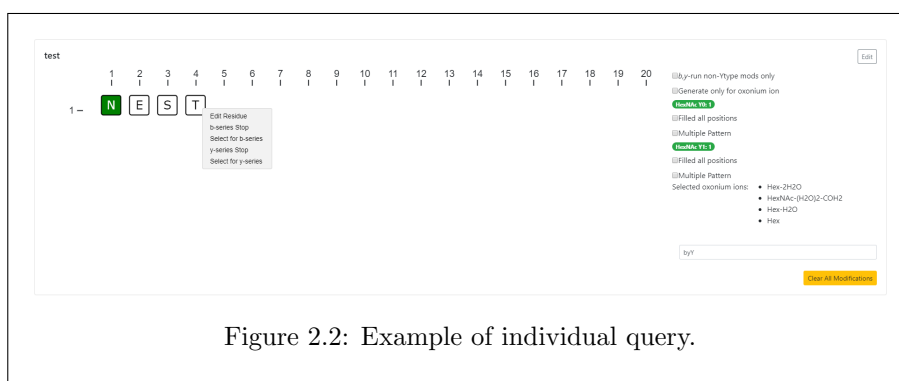


Figure 2.2: Example of individual query.

Each individual query panel is a graphical representation of the supplied sequence with modification from main settings applied to them. A context menu can be activate upon right clicking on each amino acid within the sequence with the following options

- **Edit Residue**, allow direct customization with adding and removal of any modifications to the selected residue.
- **b-series Stop**, set this residue and all after it to not being generated for  $b$  transition
- **Selected for b-series**, select this residue for b-series generation.



- **y-series Stop**, set this residue and all before it to not being generated for  $y$  transition
- **Selected for y-series**, select this residue for y-series generation.

For variable modifications, the users can also select whether or not that they want the modifications to be either fill all sites or none (for non Ytype variable modifications) by checking the **Filled all positions** box. The box after that was for all possible modifications settings. If enable, it would generate all possible combination of this modifications across different identified modification sites. Depending on the complexity of the modifications and the length of the protein, if the  $b$  and  $y$  library can become quite large and it would also take a large amount of time to create.

The checkbox **b,y-run non-Ytype mods only** would disable Ytype modification mass from calculation of ion  $m/z$  for  $b$  and  $y$  transitions for this query.

If **Generate only for oxonium ion** is checked, only oxonium ions would be included for each unique precursor at each unique RT and SWATH window combinations.

### Queries Submission

After application of all desired parameters, the user can submit the result by clicking the **Submit** button at the bottom right of the GUI. Upon triggering, each individual query would be submitted separately to the back-end and a progress bar would also appear under each query. If multiple back-end connections are available, the GUI would spread the query out to different accessible connection to ensure a balanced workload.

## Chapter 3

# Back-end

The Back-end was built on the single threaded Python asynchronous web framework Tornado exposed at `/api/swathlib/upload/` . The exposed end point has two HTTP methods enabled, GET for checking accessibility and, OPTIONS and PUT for accepting user submission. Default setting allowed connection from all origins with the following headers *Origin*, *X-Requested-With*, *Content-Type*, *Accept*.

### 3.1 User Query Preprocessing

Upon receiving the query, SWATHLib would attempt to resolve any possible conflict of variable modifications by recursively generate all combination of them at the conflicted positions. If different Ytype transitions are included, only one Ytype transition can exist at the same time. For each combination of modifications, two modifications dictionaries would be generated, one for all static modifications in the combination, the other for all variable modifications in the combination.

Using the dictionaries containing information modifications' positions, the program, in case of multiple modification pattern, would attempt to again recursively iterate through all combination of positions with and without presence of the modifications to generate unique multiple variable modifications modification pattern.

### 3.2 Theoretical Ion Generation

## Chapter 4

# Common Usage Scenarios