VIB Department of Medical Protein Research, UGent

Department of Biochemistry

Peptizer

a tool for assessing false positive peptide identifications and manually validating selected results

Instruction Manual

by

Kenny Helsens

Co-authors: Prof. Dr. K. Gevaert, Prof. Dr. J. Vandekerckhove, Dr. L. Martens

2008

Contents

Abbreviations

ARFF	Attribute Relation File Format
CSV	Comma Separated Values
MIS	Mascot Ion Score
MIT	Mascot Identity Threshold
MS/MS	Tandem Mass Spectrometry

Chapter 1

Introduction

Large datasets generated by **MS/MS-driven proteomics** are most commonly analyzed by algorithms such as **MASCOT**, X!Tandem and SEQUEST. These perform a comprehensive sequence database search and thereby suggesting multiple peptide hypothesises for each MS/MS spectrum. Typically, the highest scoring peptide hypothesis with an e-value lower then 0.05 is accepted. However, the difficulty is to verify **whether that suggested peptide hypothesis is correct**. By post-processing database search results, an **extra layer of validation** can be added to these peptide hypothesises. **Peptizer** was developed as a **configurable post-processing** platform that can be applied to **separate suspicious peptide hypothesises** from valid ones, and subjecting the former to manual validation.

Instead of verifying peptide hypothesises using fixed assumptions, Peptizer makes use of so-called **pluggable assumptions**. These are called **Agents** and can vote on a specific property of a peptide hypothesis. By combining a group of Agents, their votes are aggregated and all together form a profile that separates peptide hypothesises in .

Peptizer presents the selected peptide identifications in an **extensive manual validation environment**. Therein, general and specific information, together with spectrum-derived information place a critical scientist in an information-rich and therefore **optimal position to validate** database search results.

Chapter 2

Installing Peptizer

Installing Peptizer is a two-step process: first, download and install Java version 1.5 or higher and second, download the Peptizer files from the project website at http://genesis.ugent.be/peptizer.

2.1 Downloading Java

As Peptizer is a Java application, Java must be installed properly. Check this by opening the console

Windows - *Windows Start// Run // cmd*
Unix - *Open the shell of the distribution*

and type

java -version

Now something like

java version "1.6.0_03"

should appear which indicates that Java version 1.6 is installed properly. If Java version 1.5 or later is not installed, then go to http://java.com/ and follow the instructions to install the latest Java version.

2.2 Downloading the Peptizer files

There are three types of Peptizer files:

1. Java libraries
2. Configuration files
3. Start-up script

You can install these files properly by the Java installer you find at the project website (http://genesis.ugent.be/peptizer/download/installer.html).
Start Peptizer through *double clicking* the start-up script.

Chapter 3

Using Peptizer

This chapter explains the basics of using Peptizer. We will here explain how to **start a new Peptizer** task and how to **handle or save results** from such a task. Furthermore, we will indicate how to **distribute Peptizer data** and how to manage the **configuration files**.

Once Peptizer has been set up as described in chapter 2, start Peptizer by the *peptizer.bat* (or *peptizer.sh* in a UNIX environment) script. Upon launching the script, a window will appear from which Peptizer can be started (3.1).

3.1 Starting a Peptizer task

Start a new task from the menu bar at the top (figure 3.1).

Two types of tasks can be distinguished. The first and most important is **the selection task**. Here, the peptide hypothesises that were selected by the profile, are handled in the manual validation interface of Peptizer. A second task is **the ARFF task**. Here, the peptide hypothesises are handled into an ARFF file. This file contains all peptide hypothesises labelled whether they did or didn't match the profile along a feature vector with the Agent inspection results or votes.

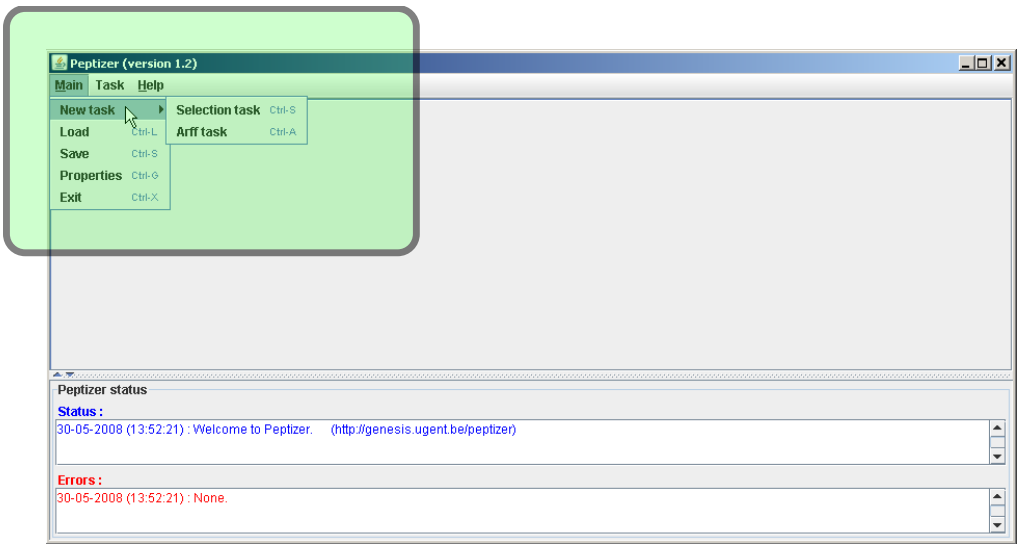


Figure 3.1: The start-up view of Peptizer. The top panel displays the menu bar wherein a new Peptizer Tasks can be started. The bottom panel displays a status panel with a welcoming message. The status panel will be updated upon user actions.

3.1.1 Selection task

Overview

Start a new selection task by pressing `Ctrl-T` or by *clicking* in the menu bar at the top:

Main // New Task // Selection Task

A new dialog appears wherein a new task can be created(3.2).
Therein, the important parts of the dialog are labelled by characters. Just below the figure, a short explanation is shown.

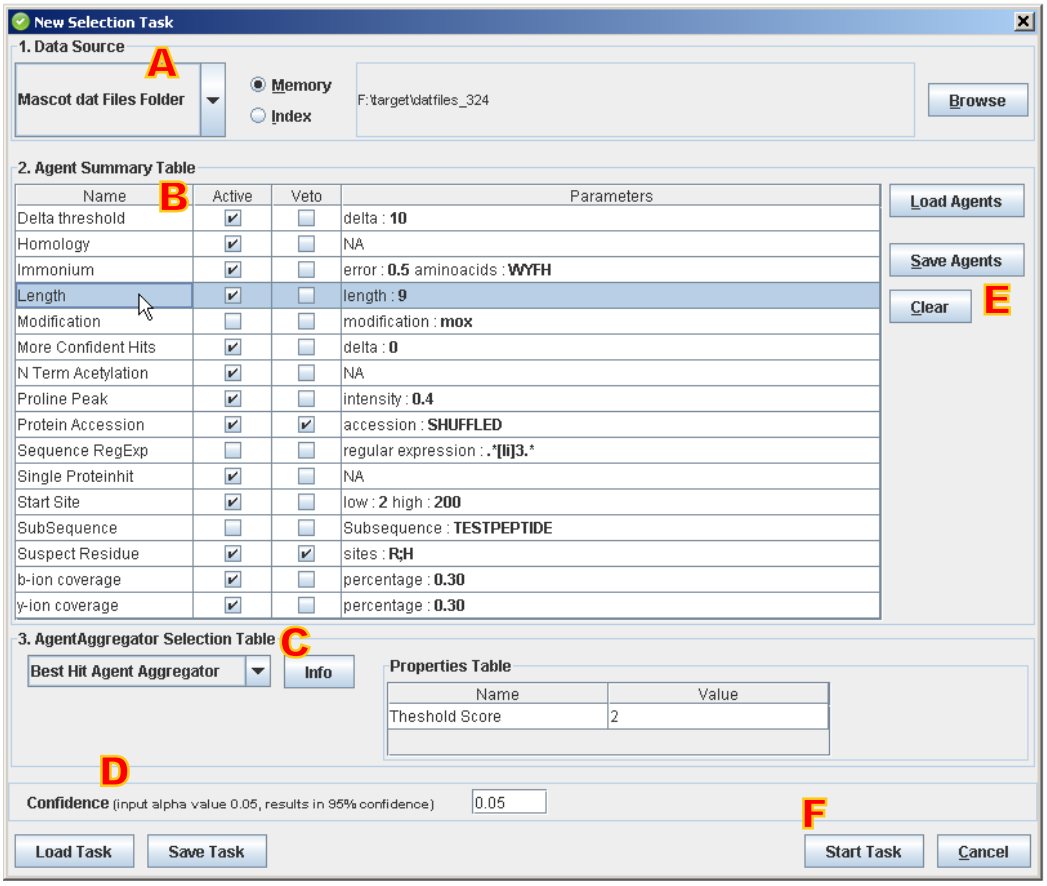


Figure 3.2: Overview of the new selection task dialog. The data source is defined in A. The Agents are activated and customized in B. The AgentAggregator is selected and customized in C. The Mascot confidence level for peptide hypothesises is defined in D. Agent configuration files can be cleared, stored or loaded in E and, finally, a new task can be started in F.

A. Defining the data source.
Which MS/MS spectrum identifications must be processed?

B. Activating and customizing Agents.
What rules do the peptide identifications have to fit?

C. Selecting and customizing the AgentAggregator.
How should the Agent votes be aggregated to judge a peptide hypothesis?

D. Defining the Mascot probability.
When is a peptide hypothesis considered confident?

E. Storing, clearing or loading a Peptizer configuration.
What pre-defined configuration must be used?

F. *Start the selection task!*

Each of these parts are explained into more detail below.

Defining the data source

The data source relates to the peptide hypothesises that must be inspected. Peptizer currently only inspects peptide hypothesises from Mascot result files. These files are stored at the Mascot server and contain the core results of the interpreted MS/MS data. Different data sources can be defined in Peptizer from the pull-down menu.

Mascot dat Files Folder Select a folder using the *browse* button. Peptizer will then process all the peptide hypothesises in all Mascot result files in that selected folder (other files in that folder will be ignored).

Mascot dat File Select a Mascot result file by the *browse* button. Peptizer will then process all the peptide hypothesises in the selected Mascot result file.

ms_lims project Create a ms_lims database connection using the *create connection* button.[1] Once a connection is established, select a ms_lims project from the pull-down menu. Peptizer will then process all the peptide hypothesises in the ms_lims project.

ms_lims identification id After a connection to a ms_lims database has been established, enter one ms_lims identification id per line in the textarea. Peptizer will then process only the MS/MS spectra and their peptide hypothesis of the given identification ids.

The Mascot result files can be parsed in two different ways. Select the appropriate radio button:

Memory uses in-memory parsing. This is fastest though very memory intensive. Use this option for files up to tens of megabytes.

[1] ms_lims stands for Mass Spectrometry oriented LIMS system (more information can be found at http://genesis.ugent.be/ms_lims/), which organizes raw MS/MS data and their interpretation by Mascot in a project-wise fashion.

Index uses index-based parsing. This is slower however far less greedy on virtual memory. Use this option for files above 100MB.

Activating and customizing the Agents

Each Agent inspects a property of a peptide hypothesis and casts a vote on each element coming from the data source. This vote has either an approving, reserving of rejecting opinion for the final grouping of peptide hypothesises. This table serves to select one or more Agents altogether having the properties of the peptide hypothesises that will be merged.

Each table row represents an Agent and each table column relates to another function in Peptizer:

Name Very short identifier of an Agent serving as a row header.

Active Simple conditional whether this Agent should be used or not. Check this checkbox if an Agents inspection must be enabled.

Veto Extra rights for an Agent vote. If checked, a peptide hypothesis will always be selected when this Agent approves selection.

Parameters Optional settings for an Agent. Some Agents require an additional parameter, this is mostly a user-defined threshold used by the Agent during inspection. If the parameter says 'NA', no parameters are to be set. Else, the table shows the current settings. These can be modified by *double-clicking* the parameters.

> HINT
> A short tooltip describing an Agents logic can be shown by *holding* the mouse cursor on its name cell.

Selecting and customizing the AgentAggregator

The AgentAggregator aggregates the votes from the different Agents. As there are different logical paths for aggregation, there are different AgentAggregators. Select an appropriate AgentAggregator by *clicking* the pull-down menu on the left. By *clicking* the info button a dialog will pop up describing the selected AgentAggregator. This dialog informs on the

general concept, the logic of the aggregation and the optional settings. An example dialog for the BestHitAgentAggregator is shown in figure 3.3.[2] Modify these optional settings by *double-clicking* the value cells.

Defining the Mascot confidence

The Mascot Identity Threshold (MIT) is calculated for a given alpha value. This alpha value reflects the probability that a peptide hypothesis is random.
Insert an appropriate alpha value for MIT calculation in the textfield at the bottom. Only peptide hypothesises with a Mascot Ion Score (MIS) higher then the MIT will be considered by Peptizer. Note that secondary and tertiary ranked peptide hypothesises with a MIS > MIT will be considered by Peptizer.

Storing, clearing or loading a Peptizer configuration

Once a set of Agents is selected and customized, that configuration can be saved into an Agent configuration file. These files can be re-loaded at a future time or at another place. Save or load an Agent configuration file by *clicking* the Save Agents button (`Alt-S`) or the Load Agents button (`Alt-L`). By *clicking* the clear button (`Alt-C`), the table will be reset for a clean start.

Find out more on the configuration files in section 3.5.

3.1.2 ARFF task

> "An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software."
> Quotation from http://www.cs.waikato.ac.nz/~ml/weka/arff.html.

The Arff task is similar to the Selection Task as Agents and AgentAggregators are similarly selected and customized. The difference lies in the output presentation to the user. Instead of presenting the selected peptide hypothesises in the manual validation interface, a single

[2] As we cannot document custom built AgentAggregators in this manual, we opted for documentation by the configuration files that can also be viewed in real-time. We therefore refer to Peptizer for documentation on the different types of AgentAggregators.

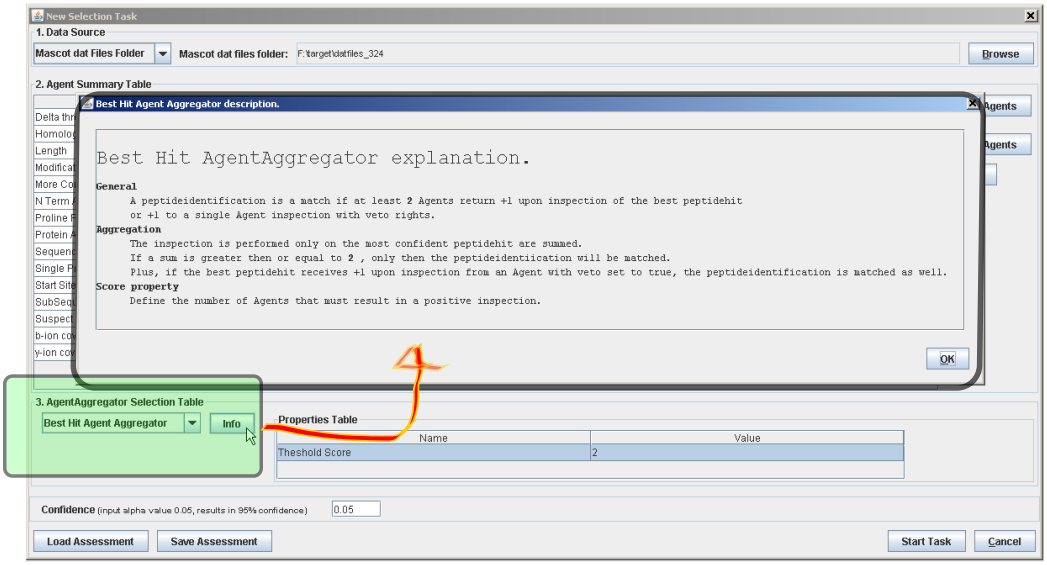


Figure 3.3: The AgentAggregator information dialog. By clicking the info button, a description dialog informs on the general concept, the logic of the aggregation and the optional settings of the selected AgentAggregator.

ARFF file is produced containing all the peptide hypothesises. For each peptide hypothesis, Peptizer creates a feature vector and a single label. One feature relates to one Agent inspection and the label corresponds with the aggregation result.

Overview

Start a new Arff task by by pressing `Ctrl-A` or by *clicking* the menu bar in the top:

Main // New Task // Arff Task

In the Arff Task dialog two types of information can be chosen to fill the feature vector. Choose by *clicking* on one of the following radio buttons:

Detailed result The feature is the peptide hypothesis variable used upon inspection.
e.g. *'6' will be the feature value upon inspection of a 6 amino acid long peptide by the Length Agent requiring a peptide hypothesis to be longer then 8 amino acids.*

Match result The feature value is the Agent vote after inspection.
e.g. *'1' will be the feature value upon inspection of a 6 amino acid long peptide by the Length Agent requiring a peptide hypothesis to be longer then 8 amino acids.*

The main purpose of this task type is to show that different tasks can be created with the common concept of Agents and AgentAggregators. Peptizer itself is focussed at the level of a single peptide hypothesis. Therefore, we directed this task specifically towards more advanced data-analysis at the level of groups of peptide hypothesises.

Besides the WEKA library, Arff files produced by Peptizer can be interpreted by free initiatives such as **Rapidminer**[3] or the **R/Weka package** for the R[4] project of statistical computing.

3.2 Handling a Peptizer task

As manual validation is a tricky and time-consuming job, this should be facilitated as much as possible. First, the user only validates those peptide hypothesises selected by set criteria. As such, no time is spend on peptide hypothesises that are not of interest. Second, the user is placed in an **optimal environment** to make a final decision on the validity of a peptide hypothesis. In such an environment, the user has a compact view with **general information** and **spectral information** along with **Agent-driven pluggable information**. These are core ideas of the Peptizer platform, both assisting the manual validation process and improving the specificity of the results.

After a selection task is finished, the selected peptide hypothesises are shown in the manual validation interface of Peptizer. The following section explains how to handle the task results in Peptizer's manual validation interface.

3.2.1 Overview

The main functions of the manual validation interface of Peptizer are shown in figure 3.4.

The peptide hypothesises selected by the criteria defined by Agents and AgentAggregators are listed in the **Identification Tree**. The tree header displays the size and the overall

[3]RapidMiner is an open-source data mining solution combining both leading-edge technologies and a broad functional range. Applications of Rapidminer cover a wide range of real-world data mining tasks. More information on the Rapidminer community edition can be found at
http://rapid-i.com/content/blogcategory/10/69/lang,en/

[4]R is a free software environment for statistical computing and graphics.
More information on R can be found at http://www.r-project.org/

validation status. Each spectrum node in the tree is an identified MS/MS spectrum and unfolds peptide hypothesises from this MS/MS spectrum as tree leaves. Also, the tree can be filtered by validation status or by Agent vote.

By *double-clicking* a spectrum node, a new **MS/MS spectrum identification** tab is added into the central panel. Therein, the suggested peptide sequence and MS/MS spectrum annotated by fragmentation information is presented. In the bottom of a MS/MS spectrum identification tab an information table is shown with both general and Agent information are shown. The MS/MS spectrum identification can be validated by the buttons on the bottom right edge.

Finally, in the very bottom, the **Status Pane** logs actions performed in Peptizer.

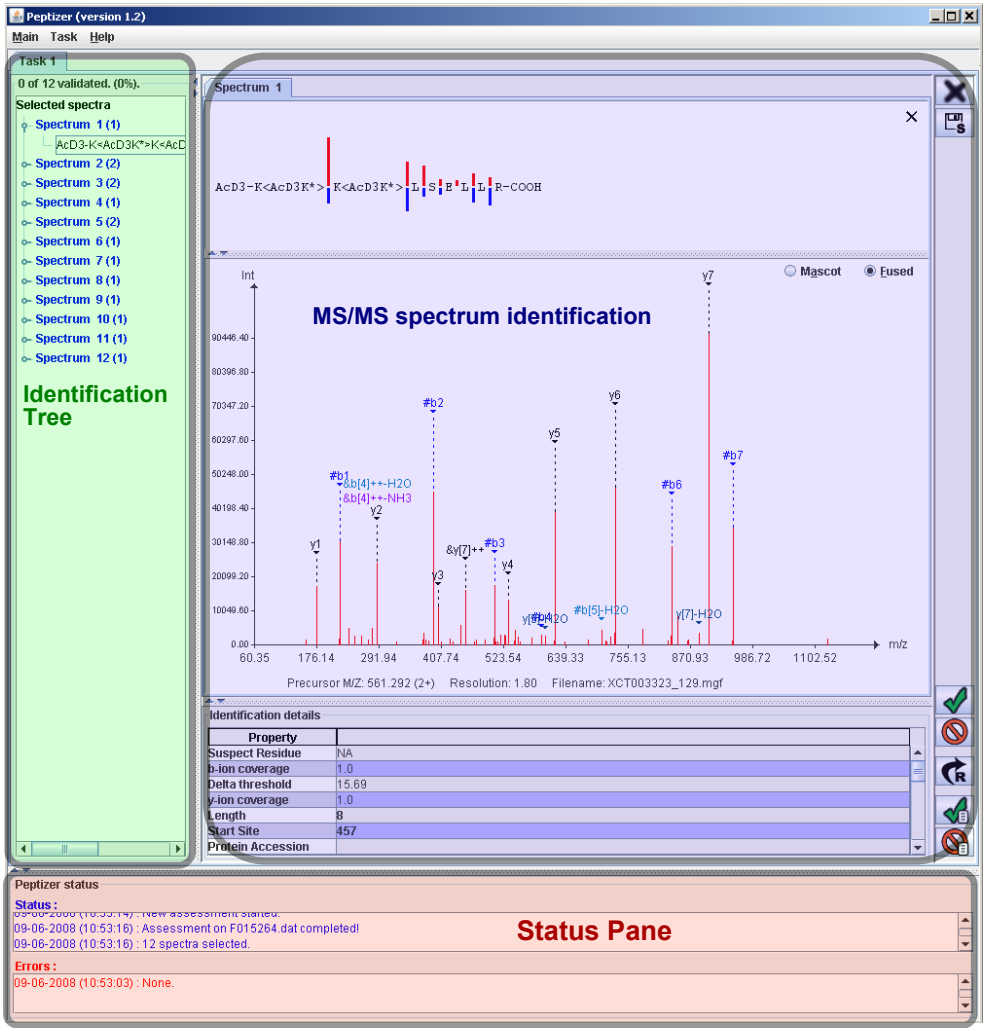


Figure 3.4: An overview of the main functions of the Peptizer validation gui. On the left side is the Identification Tree with the selected MS/MS spectra. By double clicking a spectrum tree node, a MS/MS spectrum identification tab of that spectrum is added in the central panel. In the bottom, the Status Pane is updated upon a variety of actions.

3.2.2 Identification tree

The identification tree is shown in figure 3.5

The Identification Tree header displays a number summary of the validation status. In figure 3.5, the header `"0 of 12 validated. (0%)"` means that none of the 12 selected MS/MS spectra was validated. Each of these 12 spectra are presented as spectrum nodes. By *double-clicking* a spectrum node, a new MS/MS spectrum identification tab is added in the center panel (see 3.2.3). Such a spectrum node has the folowing structure:

`Spectrum list identifier (Number of confident peptide hypothesises)`

The spectrum node displays the number of confident peptide hypothesises, which reside as tree leaves beyond each spectrum node. For example `"Spectrum 5 (2)"` in figure 3.5 indicates that there were 2 confident peptide hypothesises for MS/MS spectrum 5. Such a peptide hypothesis leaf has the following structure:

`Modified peptide sequence | Mascot Ionscore (Mascot Identity Threshold)`

Upon validating a MS/MS spectrum, the spectrum node colours green or red when respectively accepting or rejecting the spectrum node. Also note that the tree header updates the validation status.

A filter can be applied on the Identification Tree, which facilitates navigation through larger trees.

Apply a filter by *clicking* the menu bar in the top:

Task // Filter // ...

Disable one or both filters by *clicking* the menu bar in the top:

Task // Filter // Disable filters

The effect of two different filters is shown in figure 3.6.

Agent filter shows MS/MS spectrum identifications by Agent votes.

Validation filter shows MS/MS spectrum identifications that have not been validated yet.

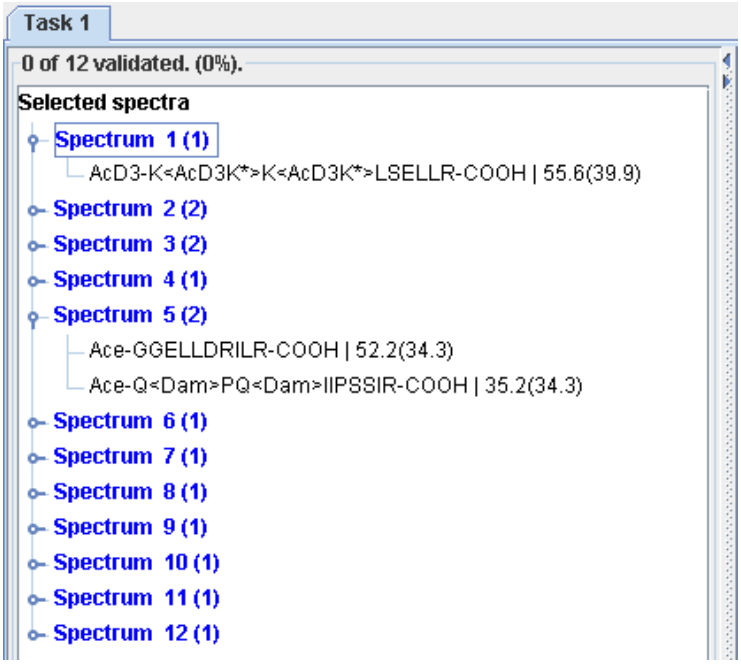


Figure 3.5: The Identification Tree lists the selected MS/MS spectra. The tree header displays the size and the validation status. The tree unfolds first with spectrum nodes and then with peptide hypothesis nodes. By *double-clicking* the spectrum node, its peptide hypothesises are presented in detail in another panel.

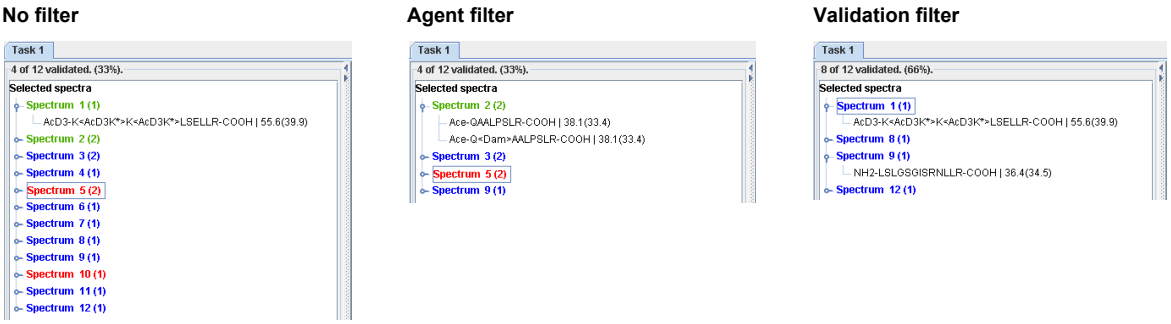


Figure 3.6: The Identification Tree filters. On the left, no filters are applied. In the middle, the Agent filter was applied and therefore only the MS/MS spectrum identifications with a selective vote for that Agent are shown. On the right, the validation filter was applied and therefore all validated MS/MS spectrum identifications are now hidden.

3.2.3 MS/MS spectrum identification tab

The central panel displays multiple tabs. Each tab is a MS/MS spectrum identification tab showing all details on one MS/MS spectrum and its proposed peptide hypothesis(es). These peptide hypothesises are theoretically fragmented by Peptizer to align them with the MS/MS spectrum. This yields **fragmentation information** that is combined with **general information**. The MS/MS spectrum identification tab also shows **pluggable information** driven by Agent inspections. Finally, an MS/MS spectrum identification can be validated here.

These different types of information are illustrated below(figure 3.7).

A MS/MS spectrum identification can be closed in the pop-up menu by *right-clicking* the tab or by *clicking* the small 'x' button on the top-right. All spectrum tabs can be cleared by *clicking* the large 'X' on the top-right.

Fragmentation information: The annotated MS/MS spectrum

The MS/MS spectrum is stored in the Mascot result files. This is used to draw the MS/MS spectrum on a custom Spectrumpanel component. The X-axis and Y-axis respectively plot the mass over charge and intensity values of the fragment ions. The user can interpret the fragmentation spectrum by the following actions:

Left click in the spectrum and drag a mass over charge interval to zoom into that range.

Right click zooms out to the complete MS/MS spectrum.

Hoovering the cursor over a peak shows the mass over charge value for that peak.

Left click on a peak starts peak-picking. If the cursor is moved on top of another peak, the mass over charge difference of those peaks is shown. By clicking another peak, this difference in mass is fixed and peak picking can be continued. Note that if this mass difference corresponds to an amino acid or a modified amino acid, the mass difference is annotated by that component.

Ctrl + Left click resets the primary peak picking.

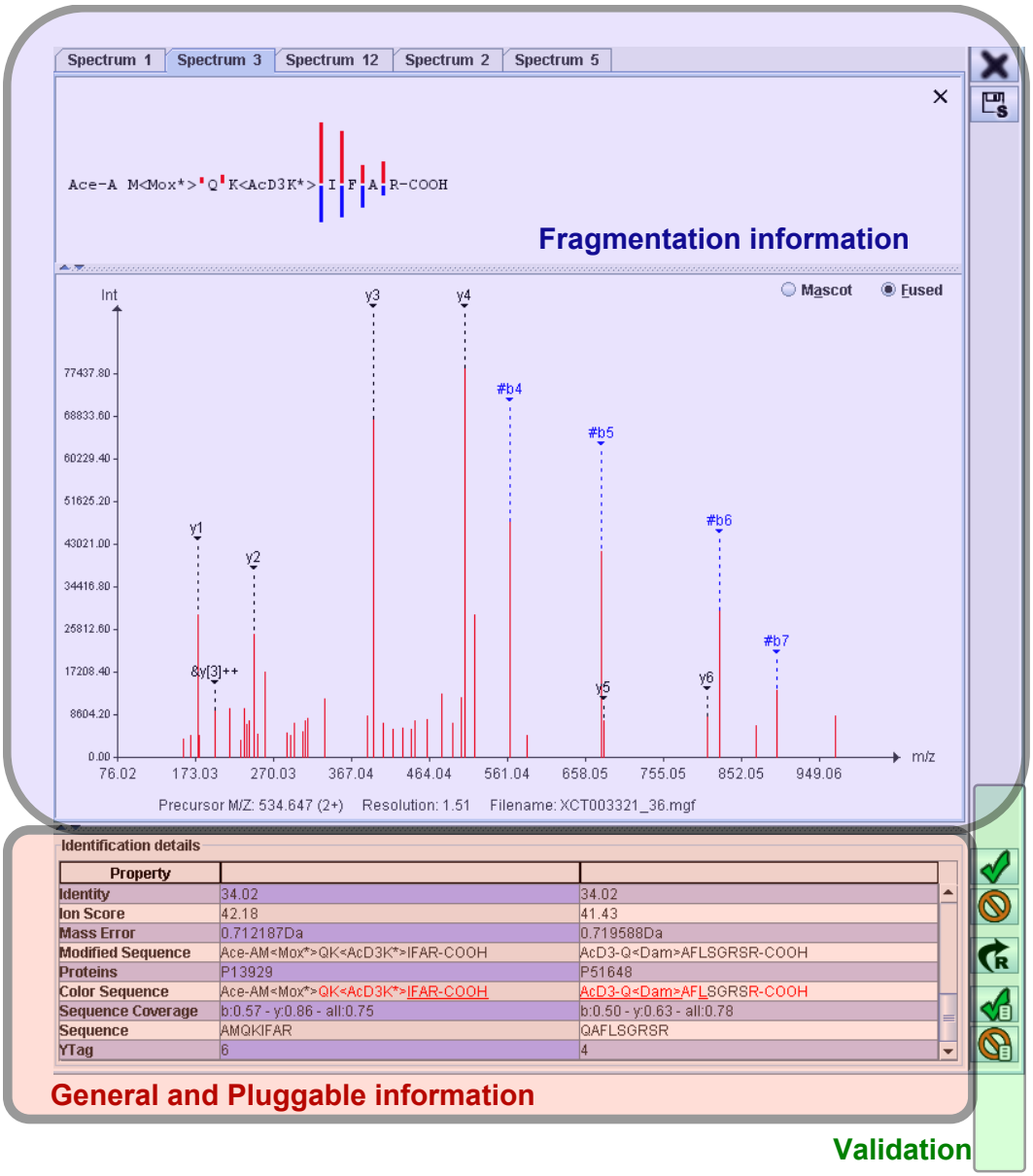


Figure 3.7: Each MS/MS spectrum identification tab displays information on a single MS/MS spectrum and its confident peptide hypothesis(es). In this tab, Peptizer combines fragmentation, general and pluggable information along with validations.

Alt + Left click holds the primary peak picking in a red font and starts a secondary peak-picking.

As Peptizer simulates the theoretical fragmentation of the peptide hypothesis, it can annotate MS/MS spectra.

For better understanding, the Mascot "**ionseries**" variable must first be explained. Mascot scores a peptide hypothesis for a MS/MS spectrum by the evidence found for a specific type of fragment ions. Two different types of fragment ions can for example be b-ions and y-ions, while double charged b++-ions and y++-ions are two other types. The different types of fragment ions are listed in the table below.

Ion Type	**Neutral Mr**
a	[N]+[M]-CHO
a*	a-NH_3
a°	a-H_2O
b	[N]+[M]-H
b*	b-NH_3
b°	b-H_2O
c	[N]+[M]+NH_2
d	a - partial side chain
v	y - complete side chain
w	z - partial side chain
x	[C]+[M]+CO-H
y	[C]+[M]+H
y*	y-NH_3
y°	y-H_2O
z	[C]+[M]-NH_2

The fragment ion masses can be calculated when [N] is the molecular mass of the neutral N-terminal group, [C] is the molecular mass of the neutral C-terminal group and [M] is molecular mass of the neutral amino acid residues.[5]

The ionseries variable comes with each peptide hypothesis and lists the ion types that Mascot considered relevant to propose this peptide hypothesis. Opposite to the Mascot HTML result page, this valuable information is used when annotating the MS/MS spectrum. This is illustrated by an annotated MS/MS spectrum shown in figure 3.8.

The relevance of the ion type is encoded by the prefix:

[5]More information on the different types of ionseries can be found on-line at the Mascot help pages. http://www.matrixscience.com/help/fragmentation_help.html

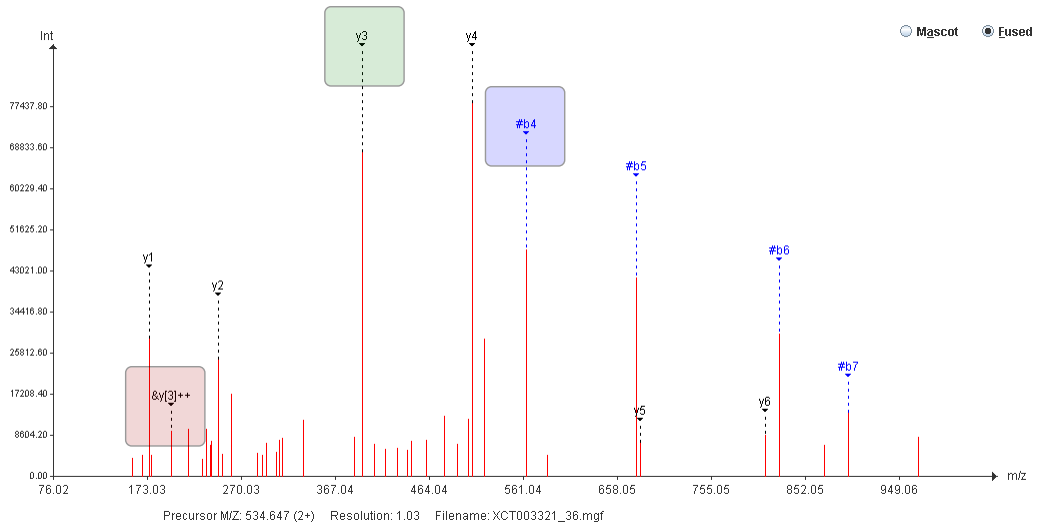


Figure 3.8: The MS/MS spectrum annotated by relevant fragment ion types.

plain y3 means that Mascot found y-ions significant and used these to score the peptide hypothesis.

hash #b4 means that Mascot found b-ions significant, however they were not used to score the peptide hypothesis.

ampercent &y++3 means that Mascot found double charged y-ions not significant and they were not used to score the peptide hypothesis.

Finally, the Mascot and Fused annotation types can now also be explained.

Mascot annotations show those fragment ion types significant for Mascot. Moreover, only the most relevant peaks are annotated.[6]

Fused annotation show the Mascot annotations plus immonium ions and the non-significant ion types (those prefixed with '&'). Also, all peaks are annotated if they are more intense then 10% of the most intense peak.

Swap between both types by clicking the corresponding radio button at the top of the panel.

[6]The most relevant peaks by the Mascot annotation are defined by the PeaksUsedFromIons1 variable of a peptide hypothesis. This is the number of peaks that were used to propose the peptide hypothesis.

As such, one can briefly inspect the MS/MS spectrum and its peptide hypothesises, as suggested by Mascot and supplemented with additionla fragment ion data.

Fragmentation information: The annotated peptide sequence

After simulating the theoretical fragmentation of the peptide hypothesis, Peptizer aligns this fragmentation information with the peptide sequence. This type of presentation was inspired by GPMDB.[7]

An illustration of an annotated sequence is shown in figure 3.9.

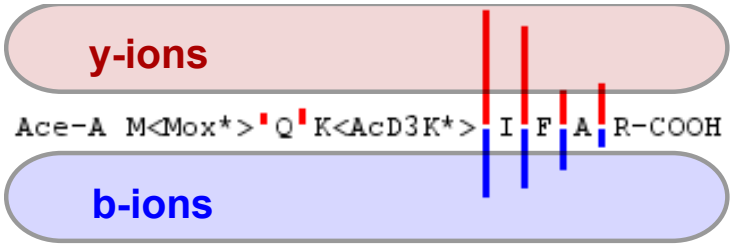


Figure 3.9: The peptide sequence annotated by b and y fragment ions.

The image is interpreted as follows. If the peptide bond fragmented and led to a b or b++ ion, a blue bar is shown below that peptide bond. Else, if the peptide bond fragmented and lead to a y or y++ ion, a red bar is shown above that peptide bond. All bars are relative to the peak intensity, where the most intense annotated peak in the MS/MS spectrum is set as the maximum bar.

The fragmentation annotation on the MS/MS spectrum and the peptide sequence are complementary. This is illustrated in figure 3.10. Here, the y1 and b7 ion are each on one side of the MS/MS spectrum, they do however originate from fragmentation of the same peptide bond. This cannot easily be seen in the annotated MS/MS spectrum since it is based on mass over charge values. Moreover, in this example, the annotated sequence shows that the C-terminal end of the peptide fragmented well and further indicates which peptide bonds broke most efficiently. Still, the annotated MS/MS spectrum is indispensable

[7]The Global Proteome Machine Database was constructed to utilize the information obtained by GPM servers to aid in the difficult process of validating MS/MS spectrum identifications as well as protein coverage patterns. http://gpmdb.thegpm.org/

as for instance the spectrum quality and presence of intense but non-annotated peaks are important observations.

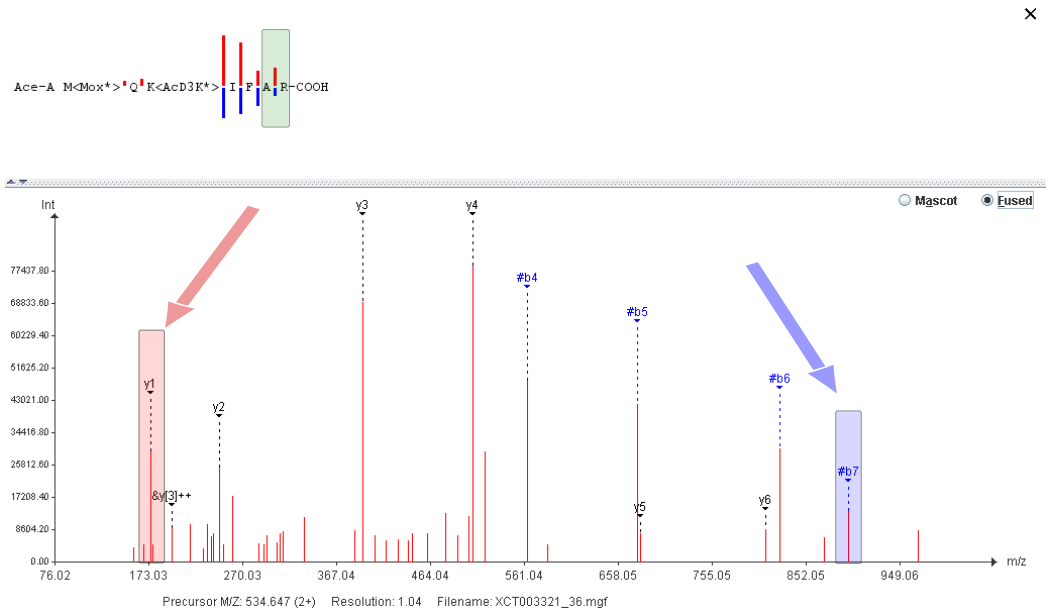


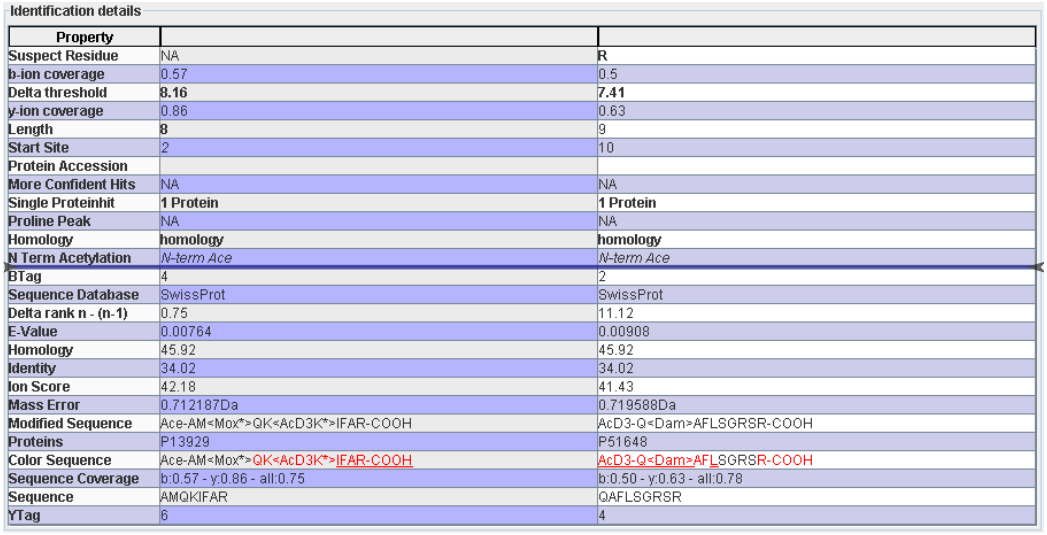
Figure 3.10: The annotation on the peptide sequence in the top and on the MS/MS spectrum in the bottom are complementary. While the peptide sequence annotation is based on peptide bonds, the MS/MS spectrum annotation is based on mass over charge values.

General and Pluggable information

The bottom of the MS/MS spectrum identification tab contains an information-rich table on all confident peptide hypothesises. Both general and pluggable information are combined in this table.

An example is shown in figure 3.11.

The first column lists the row identifiers. The other columns represent a confident peptide hypothesis, ordered according to decreasing confidence from left to right. By default, the most confident peptide hypothesis is focused by Peptizer. If there is more then one confident peptide hypothesis, right-click to select the appropriate column to focus on. The selected peptide hypothesis annotates the MS/MS spectrum and the sequence image. Moreover,



Identification details

Property		
Suspect Residue	NA	R
b-ion coverage	0.57	0.5
Delta threshold	8.16	7.41
y-ion coverage	0.86	0.63
Length	8	9
Start Site	2	10
Protein Accession		
More Confident Hits	NA	NA
Single Proteinhit	1 Protein	1 Protein
Proline Peak	NA	NA
Homology	homology	homology
N Term Acetylation	*N-term Ace*	*N-term Ace*
BTag	4	2
Sequence Database	SwissProt	SwissProt
Delta rank n - (n-1)	0.75	11.12
E-Value	0.00764	0.00908
Homology	45.92	45.92
Identity	34.02	34.02
Ion Score	42.18	41.43
Mass Error	0.712187Da	0.719588Da
Modified Sequence	Ace-AM<Mox*>QK<AcD3K*>IFAR-COOH	AcD3-Q<Dam>AFLSGRSR-COOH
Proteins	P13929	P51648
Color Sequence	Ace-AM<Mox*>QK<AcD3K*>IFAR-COOH	AcD3-Q<Dam>AFLSGRSR-COOH
Sequence Coverage	b:0.57 - y:0.86 - all:0.75	b:0.50 - y:0.63 - all:0.78
Sequence	AMQKIFAR	QAFLSGRSR
YTag	6	4

Figure 3.11: The information table with both general and pluggable information. Each column shows a confident peptide hypothesis, ordered according to decreasing confidence from left to right. Each row shows an information unit. The upper rows show pluggable information driven by Agent inspections and the lower rows show general information. The Agent vote is reflected by the typeface of the cell.

by accepting a peptide hypothesis during validation, it is the focussed peptide hypothesis that is selected while the other peptide hypothesises, if any, are automatically rejected.

The upper rows show pluggable information driven by Agent inspections. These cells display a peptide hypothesis property that was judged by the Agent. The cell's typeface also reflects the Agent vote.

plain when the Agent voted neutral for selection of the peptide hypothesis.

bold when the Agent voted to select the peptide hypothesis.

italics when the Agent voted *not* to select the peptide hypothesis.

As such, one look at the table directly shows why Peptizer selected a peptide hypothesis.

The lower rows show general information, which is always available. Examples are the peptide sequence, the protein source, the mass error, the MIS and the b-ion coverage.

HINT
A short tooltip describing the table row is shown by *holding* the mouse above the row.

Finally, the information table serves as a dense information scope on the peptide hypothesis. This complements the fragmentation information of the peptide hypothesis. As such, Peptizer provides an optimal environment to make the final decision on the validity of the peptide hypothesis proposed for a MS/MS spectrum.

Validation

After a peptide hypothesis has been inspected by Agents and by the user, the user either rejects or accepts the peptide hypothesis.

The validation functions are located in the right bottom part. To accept the peptide hypothesis, *click* the button with the green OK icon. To reject the peptide hypothesis, *click* the button with the red cross icon. Note that if there are multiple confident peptide hypothesises in the table, it is the focused peptide hypothesis that is validated. Also, if one is accepted, the others are automatically rejected. Both buttons (accept or reject) have an alternative button with an extra "text" mark. These buttons additionally show a dialog wherein the validation can be argued. If the validation status must be reset, click the button with the black reset sign. After validating a peptide hypothesis, its MS/MS spectrum identification tab will close before the next non-validated MS/MS spectrum identification tab opens.

3.2.4 Status pane

The status pane is always available. Simple messages are logged in the top status panel while error messages are logged in the error window.

3.3 Saving the results

The input of Peptizer consists of MS/MS spectra and their peptide hypothesises. Within Peptizer these can be inspected by multiple Agents resulting in a custom selection of MS/MS spectra, which can then be validated. Then, the output of Peptizer consists of the same MS/MS spectra and their peptide hypothesises, however these are now labelled by Agent inspections and user validation. This additional information can be saved as well. Either in a comma separated file format or in an ms_lims database.

Save a selection task by by pressing `Ctrl-S` or by *clicking* the menu bar in the top:

Main // Save

The save dialog has two main parts. The upper serves to select **which task** must be saved while the lower part serves to define **what information** must be included for each peptide hypothesis.

Saving to comma separated value file

Both the general and pluggable information shown in the table (see 3.2.3) can be written into a csv file. The save to csv dialog is shown in figure 3.12.

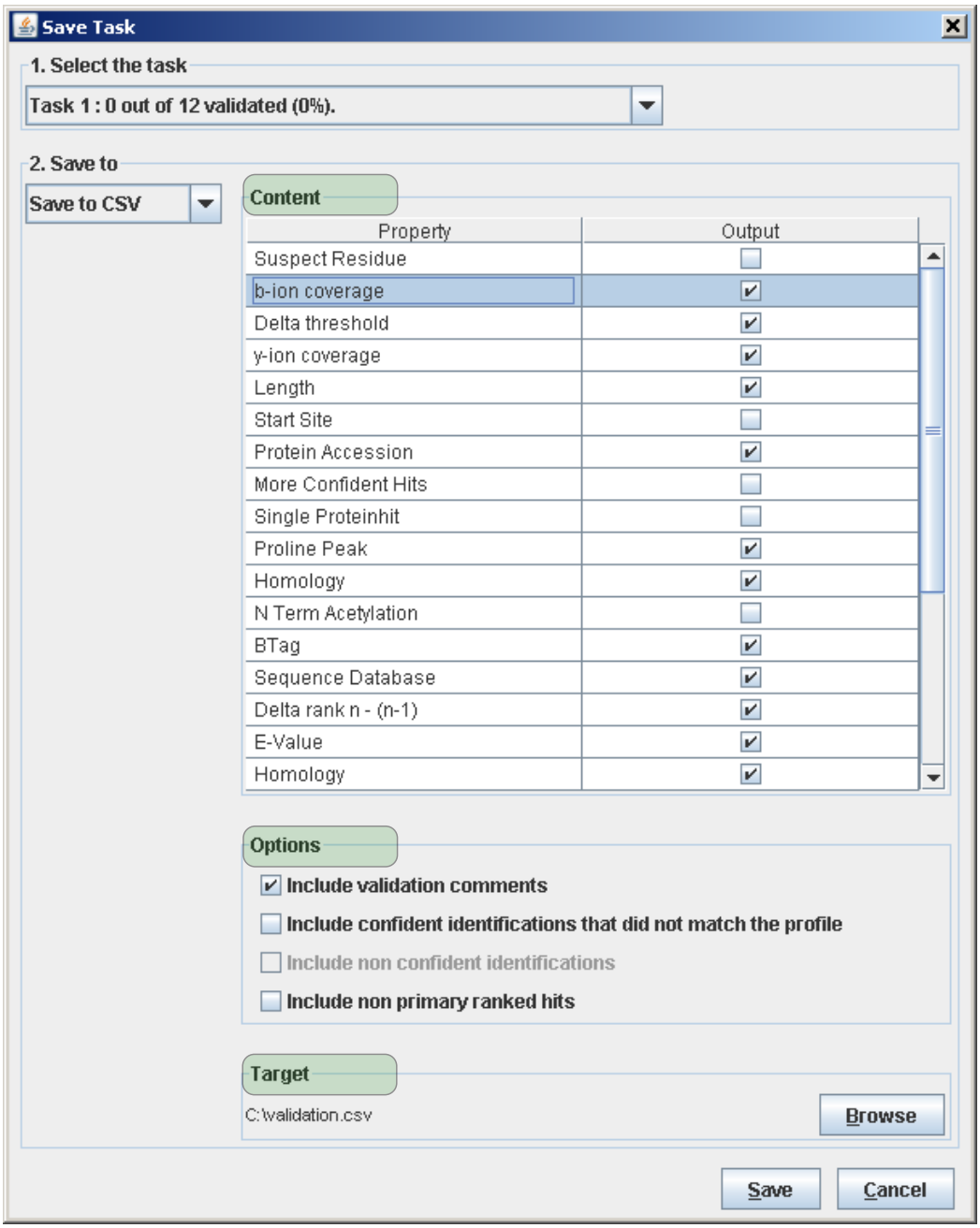


Figure 3.12: This dialog window serves to save the output into a csv file. Define the desired **content** of the csv file by *clicking* the check box aside an information cell. Below this content **options** to include other MS/MS spectra and their peptide hypothesis beside those selected by Peptizer are provided. Another kind of optional output are the user's validation comments. Finally, a **target** to save the csv file must be selected.

In this window, the user can choose the content of the csv file. Three parts can be distinguished:

Content . Select the information that must be saved. This table mirrors the information table, so some or all of the general of pluggable information can be included in the comma separated file.

Options .

Include validation comments Add the user validation comments as the last column.

Include confident identifications that did not match the profile Add confident peptide hypothesises that were not selected by Peptizer.

Include non-confident identifications Add non-confident peptide hypothesises.

Include non-primary ranked hits Add all confident peptide hypothesises, also if they are ranked second or third.

Target . Select a target file to write the output to.

Saving to PDF file

Where tab delimited files can be a starting point for further data analysis, PDF files are definitely and end point. They have a single purpose for fast data sharing and therefore still useful as email communication or web sharing. The biggest advantage of PDF files being a wide spread format and high resolution graphics. *Hence, sharing proteomics data in public repositories such as PRIDE or PeptideAtlas should be favored at all time!*

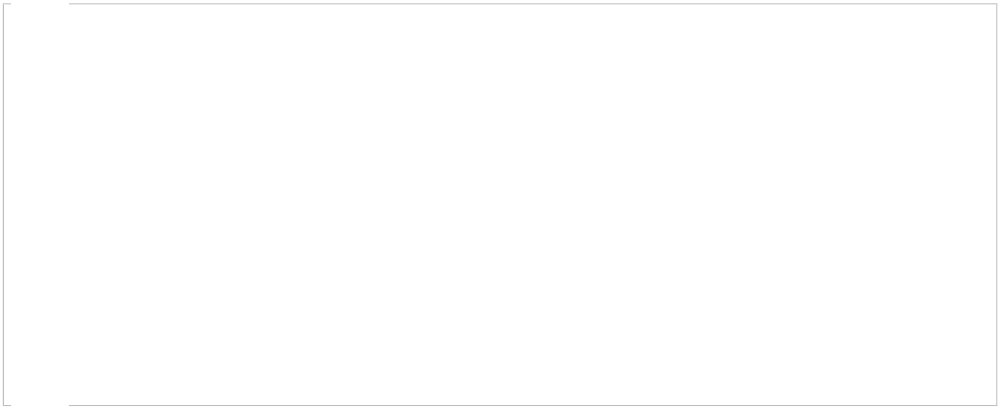


Figure 3.13: The save method for a peptideidentification in PDF format. If there were multiple peptide identifications selected by Peptizer, then a single pdf file with multiple pages is created.

Saving to ms_lims database

If the ms_lims database was used as data source, the user validation can be persisted into ms_lims. The "save to ms_lims" dialog is shown in figure **??**. At this stage, the new information concerns the user validation. Therefore, no further options are provided as all the MS/MS spectra and their peptide hypothesises are already stored in the ms_lims system.

3.4 Distributing Peptizer objects

Selected peptide hypothesises in Peptizer stand over time and space. If 250 peptide MS/MS spectra and hypothesises must be manually validated, this could last a few days. Moreover,

if doubts arise on validating a peptide hypothesis, a second opinion should be required. For this use case, Peptizer can store and load the content of the manual validation environment.

Store a task by *clicking* the menu bar in the top:

Task // Store

All stores all the MS/MS spectra.

Accepted stores the MS/MS spectra that have been accepted upon validation. (*Those that are coloured green in the tree!*)

Load a task by *clicking* the menu bar in the top:

Task // Load

Now, browse to the file that was previously stored to reload its content into Peptizer.

HINT
Go the example page on the Peptizer project web page to find such a file with a set of MS/MS spectra and validated peptide hypothesises that can be reloaded into Peptizer. This serves as an example set to start using Peptizer!
(http://genesis.ugent.be/peptizer/peptizer/download/examples.html)

3.5 Understanding the configuration files

The configuration files are essential to the plug-and-play concept of Peptizer. When Peptizer starts, these configuration files declare which Agents and AgentAggregators are available to the user in Peptizer. Thus, when custom Agents are written in Java and added to the configuration files, they will function rightaway in the Peptizer platform.

The configuration files are simple XML files and, as such, both computers and humans can easily read and modify them. They must reside in the same folder as the Peptizer start-up file. In total, there are four configuration files:

agent.xml lists the Agents that will be available to construct a voting panel and inspect peptide hypothesises.

aggregator.xml lists the AgentAggregators that will be available to aggregate the Agent inspections.

table.xml lists the general table rows that show the information in the information table.

general.xml lists general configuration properties.

Each of these are explained in the following subsections.

agent.xml

aggregator.xml

table.xml

All of these have a similar code structure. This is illustrated below by a section of the default agent.xml file.

```
<agent>
  1)  <!-- This Agent inspects the score delta between the MIS and MIT." -->
  2)  <uniqueid>be.proteomics.mat.util.agents.DeltaScore</uniqueid>
  3)  <property name="name">Delta threshold</property>
  4)  <property name="active">true</property>
  5)  <property name="veto">false</property>
  6)  <property name="delta">10</property>
</agent>
```

The outer `<agent>` tags declare the start of a **new section**. This can also be aggregator or tablerow in the aggregator.xml or table.xml file respectively. Within this section, the following lines are present:

line 1. **describes** this section in a xml comment line. Note that this is only used as a guidance. The tooltip descriptions within Peptizer are hard coded.

line 2. **identifies** this section in a unique manner. Since Java guarantees that each class has a unique class name, this is used as the unique identifier for an Agent.[8]

[8]The class name consists of the package structure and ends with the class itself. Consider it as a road to the class file whereas the track starts from "be" to "proteomics" andsoon.

line 3. **names** this section. It is the functional name used within the Peptizer GUI.

line 456. **define custom properties** for this section. Both the active and veto must have default boolean values when Peptizer starts. Moreover, the delta parameters defines the delta value (here as the required score difference between MIS and MIT) upon that Agents inspection.

Note that if there are no options for an Agent, AgentAggregator or Tablerow, no custom property lines will be shown. Else, if there are multiple options, there will also be multiple custom parameter lines.

general.xml

These are various settings for the Peptizer application. Just like the other configuration files, the values can be modified by default Peptizer values.

```
<general>
<!-- Default annotation radiobutton. -->
<property name="RDB_ANNOTATION">0</property>
<!-- Default tree height. -->
<property name="TREE_HEIGHT">20</property>
<!-- Default confidence throughout MAT. -->
<property name="DEFAULT_ALPHA">0.05</property>
. . .
</general>
```

The outer `<general>` tags line up the start of the **general properties part**.

RDB_ANNOTATION Set the default MS/MS spectrum annotation type for the GUI: 0 will show the Mascot annotations and 1 will show the Fused annotations (see 3.2.3).

TREE_HEIGHT Set the number of pixels between the tree nodes.

DEFAULT_ALPHA Set the alpha value for starting a new selection task.

ITERATOR_FOLDER_PATH Set the preset folder for the Datfile folder data source.

ITERATOR_FILE_PATH Set the preset file for the Datfile data source.

SAVEVALIDATION_CSV Set the preset file for the Datfile data source

MODIFIEDSEQUENCE_FRAGMENTATION_PANEL Set the annotations on the modified or the non-modified peptide sequence if respectively true or false.

CONNECTION_PROPERTIES Set the coordinates to establish a ms_lims database connection. This file can include the DRIVER and URL fields from the connection dialog if Peptizer is used to connect to the ms_lims database system.

FUSED_INTENSITY_PERCENTAGE Set the lower intensity threshold as a percentage in relation to the most intense fragmention up to where peaks from the MS/MS spectrum will be annotated.

ENABLE_CONFIDENT_NONSELECTED_OBJECT.OUTPUTSTREAM Set Peptizer to stream confident peptide hypothesises that were not selected into a temporary file. If set to true, an extra option appears in the SavetoCSV function to save these peptide hypothesises as well. Note that this temporary file is deleted after Peptizer is closed and this comes along with a performance cost.

ENABLE_NONCONFIDENT_OBJECT.OUTPUTSTREAM Set Peptizer to stream non-confident peptide hypothesises into a temporary file. As such, data on the negative population can easily be gathered. If set to true, an extra option appears in the SavetoCSV function to save these peptide hypothesises as well. Note that this temporary file is deleted after Peptizer is closed and this comes along with a performance cost.

Saving these allows to easily gather on the negative population of peptide hypothesises.

3.6 Using Peptizer from the command line

Basically, this means that a new Peptizer selection task can be started by a single command. Commands are run from the commands window or from the shell respectively in a windows or a UNIX environment. As this is no convenient environment for most users, running Peptizer from the command line should only be considered by informatics oriented users.

Why is this useful? By the command line, multiple Peptizer tasks can be started with each task having a different Agent configuration or a distinct data source. Also, a Peptizer selection task can be started at a central server without any graphical interaction.

This can be done in three steps:

1. **Sets the classpath** to define the classes that must be loaded into the Java virtual machine.[9] Therefore, this part defines the libraries required by Peptizer.[10]
   `set CLASSPATH=.\config;.\peptizer-1.1.jar;.\mascot_datfile-1.5.3.jar; ..`

2. **Starts a Java process** by a command. Note that here you can also supply Java Virtual Machine properties as a parameter. In the example, `-Xmx512M` reserves 512mb of memory to run this Peptizer task. Finally in this step, the Peptizer Class is supplied whose main method can start a Peptizer task in command line mode.
   `java -Xmx512m be.proteomics.mat.main.Peptizer`

3. **Sets the program parameters** to run a selection task.

```
--sourcetype file
--source C:\\Temp\\F004071.dat
--target C:\\Output\\result.csv
--table C:\\applications\\Peptizer\\table.xml
--agent C:\\applications\\Peptizer\\agent.xml
--aggregator C:\\applications\\Peptizer\\aggregator.xml
--general C:\\applications\\Peptizer\\general.xml
--parsing memory
```

Note that by running `be.proteomics.mat.main.Peptizer` without the program parameters a help text appears in the command line.

This will select all confident peptide hypothesises from `F004071.dat` that match the profile by the Agents and AgentAggregator defined in `agent.xml` and `aggregator.xml`. The

[9] Find out more on the Java classpath at http://java.sun.com/j2se/1.3/docs/tooldocs/win32/classpath.html

[10] The Java libraries required by Peptizer are listed at the project website download page http://genesis.ugent.be/peptizer/peptizer/download/main.html

probability of a confident peptide hypothesis is defined in `general.xml`. Finally, a csv file will be written as `result.csv` containing general, pluggable information and a selection label as columns.

Chapter 4

Extending Peptizer

4.1 Introduction

Upon inspecting MS/MS identification results, a critical scientist has certain **expectations**. Some of these are automatically used by the database search algorithm to identify an MS/MS spectrum while others are not applicable on a large scale and therefore not used. To the first group belong mass differences between fragment ions as these are general and informative for the peptide sequence. Database search algorithms commonly use these as **robust features** to interpret a MS/MS spectrum. To the second group belong less general features. For example, it is known that proline-containing peptides are more prone to fragment N-terminally to proline upon CID. But, since not every peptide contains a proline database search algorithms cannot use this as a general robust feature. Hence, this type of information can still be used in a **complementary** level aside the database search algorithm.

The Peptizer platform does just this. If a critical scientist has certain expectations for his or her MS/MS results, these expectations can be translated into **custom Agents**. Moreover, the methodology to group these expectations can be translated into a custom **AgentAggregator**. Both suggest extending Peptizer **to automate the inspections of a critical scientist's expectations**.

This chapter explains how to extend Peptizer by creating custom Agents and AgentAggregators.

4.2 Overview

Both the Agent and the AgentAggregator have a basic **input output** operation. This is illustrated by the figures below.

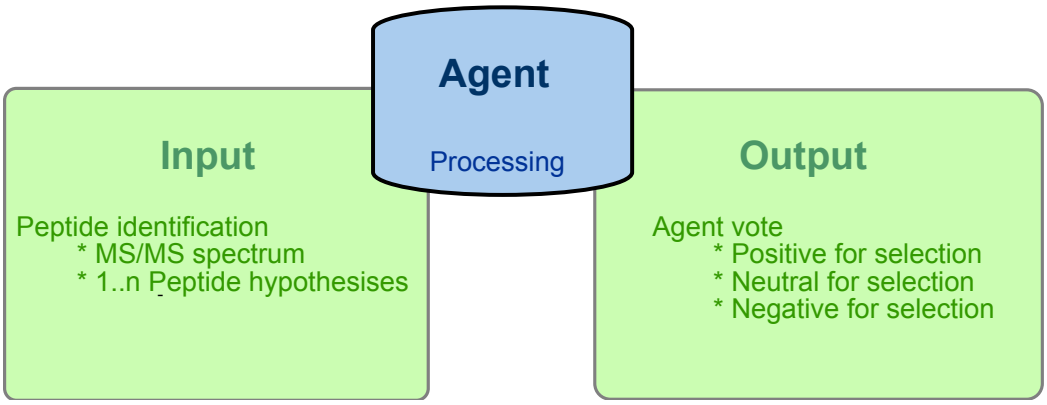


Figure 4.1: The basic input/output structure of an Agent. As input, the Agent receives a PeptideIdentification object that consists of a single MS/MS spectrum and a number of peptides hypothesises suggested for that spectrum. After this input has been processed by an Agent, a vote is casted as output. This vote refelcts the Agent's opinion for selecting or not selecting the given peptide identification.

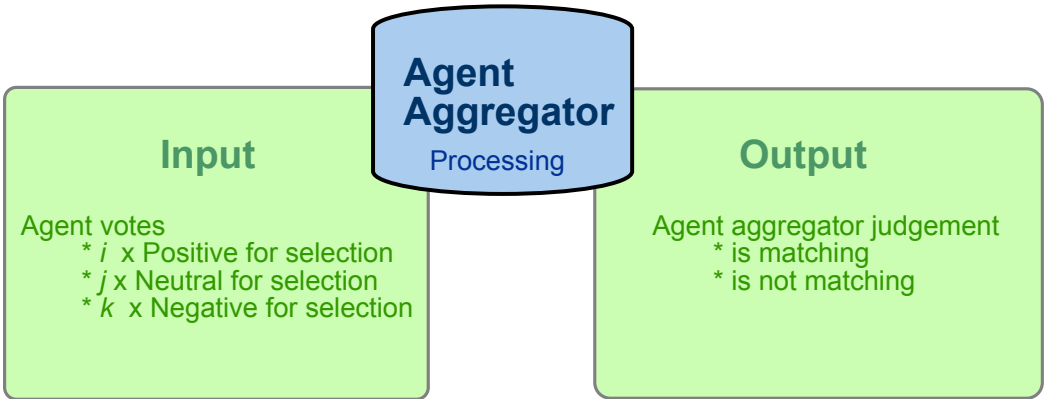


Figure 4.2: The basic input/output structure of an AgentAggregator. As input, the AgentAggregator receives a collection of Agent votes. Therein, i Agents vote for selection of a peptide identification, j Agents vote neutral for selection while k Agents vote against selection. The Agent agregator then processes all votes and concludes as output whether or not the peptide identification is matching and should therefore be selected or not.

This concept must be kept in mind when extending Peptizer. Next up are instructions for creating an **easy Agent** followed by a **more complex Agent**. Finally, the creation of an **easy AgentAggregator** is instructed as well.

4.3 Writing the first Agent

For the easy example, a length Agent will be made. Its aim is to **verify whether a peptide's length is shorter or longer then a given threshold length**. The Agent could then for example be used to select short peptides as these are more prone to generate false positive peptide identifications.

First and most important: each custom Agent must **extend the abstract Agent class**. Thereby, each Agent has **common variables** and a **common signature**. Examples for common variables are a name, a status for both the activity and the veto option as well as a common method to get a unique identifier.

Second, to set an Agent's variables from the agent.xml configuration file, an Agent must be **initialized**. After being initialized, an Agent can then receive input and produce output. The initialization is also the place to define **custom parameters** like the threshold length in this example. This is illustrated in the code snapshot below. There, the value of the LENGTH variable is used to get the appropriate value from the agent.xml configuration file.

Algorithm 1 Constructing an Agent

```
public class LengthAgent extends Agent {

   public static final String LENGTH = "length";

   public LengthAgent(){
      initialize(LENGTH);
   }
}
```

Third, the **inspect method** must be implemented from the abstract Agent class. This method reflects the basic input/output structure of an Agent: **a PeptideIdentification object is an input parameter** and **an array of AgentVotes is returned as output**

(one for each peptide hypothesis from the MS/MS spectrum). Note that the different votes are static by on the **AgentVote enum**.

The inspect method for the LengthAgent can be seen as following. Get the length threshold as a local variable from a Properties object that was created during initialization of the Agent. Then create an array to fit n Agentvotes where n equals the number of confident peptide hypothesises made for the MS/MS spectrum (so we decide to only inspect on confident peptide hypothesises). Then an AgentReport is created for each of these peptide hypothesesis, wherein results are persisted.

If the length of the peptide sequence is less then the length threshold, the LengthAgent votes positive for selection. Else it votes neutral for selection. Finaly, the reports are made and stored along the PeptideIdentification. These reports are used to display the PeptideIdentification in the Peptizer graphical user interface.

Algorithm 2 Inspect method of the Length Agent

```
public AgentVote[] inspect(PeptideIdentification aPeptideIdentification) {

        int lLength = Integer.parseInt((String) (this.iProperties.get(LENGTH)));
        AgentVote[] lVotes = new AgentVote[aPeptideIdentification.getNumberOfConfidentPeptideHits()];

        for (int i = 0; i < lVotes.length; i++) {

     PeptideHit lPeptideHit = aPeptideIdentification.getPeptideHit(i);
    AgentReport lReport = new AgentReport(getUniqueID());

            int lPeptideLength = lPeptideHit.getSequence().length();

            if (lPeptideLength < lLength) {
                lVotes[i] = AgentVote.POSITIVE_FOR_SELECTION;
            } else {
                lVotes[i] = AgentVote.NEUTRAL_FOR_SELECTION;
            }

            lReport.addReport(AgentReport.RK_RESULT, lVotes[i]);
            lReport.addReport(AgentReport.RK_TABLEDATA, new Integer(lPeptideLength));
            lReport.addReport(AgentReport.RK_ARFF, new Integer(lPeptideLength));

            aPeptideIdentification.addAgentReport(i + 1, getUniqueID(), lReport);
        }
        return lVotes;
    }
```

In the end, the LengthAgent returns an Agentvote for each confident peptide hypothe-

seis. Whereas each vote reflects the length of the peptide in relation to the given length threshold.

After this easy example, lets create an Agent with more advance processing features.

4.4 A more advanced Agent

4.4.1 Background

This section starts with a quote from a 2004 paper by Ross et al.:

> We have developed a multiplexed set of reagents for quantitative protein analysis that place isobaric mass labels at the N termini and lysine side chains of peptides in a digest mixture. The reagents are differentially isotopically labelled such that all derivatized peptides are isobaric and chromatographically indistinguishable, but yield signature or reporter ions following CID that can be used to identify and quantify individual members of the multiplex set.
> Ross, P. L. et al. (2004). "Multiplexed protein quantitation in Saccharomyces cerevisiae using amine-reactive isobaric tagging reagents." Mol Cell Proteomics 3(12): 1154-69.

Ross et al. introduced the iTRAQ™ methodology that has become a widespread tool for quantitative proteomics. If this chemistry is used, then iTRAQ™ reporter ions are expected to appear in the MS/MS spectrum. If these reporter ions appear in unequal intensities, then the corresponding peptide was differentially abundant in both samples.

This can now be defined as a case to create a new Agent:
Do iTRAQ™ reporter ions appear at different intensity?

4.4.2 Creating a custom Agent to inspect expectations

A custom Agent inspecting the appearance of iTRAQ™ reporter ions in the MS/MS spectrum will now be created. This Agent will be named as the ReporterIonAgent. Just as any other Agent in Peptizer, the ReporterIonAgent must extend the abstract Class Agent. This abstract class has common methods and variables among all Agents.[11] Examples of common features are the name, the activity or the veto status as well as the getters and

[11] Read more on abstract classes at http://java.sun.com/docs/books/tutorial/java/IandI/abstract.html

the setters to modify these variables. As such, the ReporterIonAgent must only encode its differences from other Agents.

A custom Agent like the ReporterIonAgent can be created from **the template signature of an Agent extension**. This template is availlable as the **DummyAgent** in the standard Peptizer distribution.[12]

Parameters Declare optional parameters that are used as options by this Agent.
The mass over charge values for reporter ions

Constructor Declare the instantiation method of an Agent extension.
The initiation of the super class Agent and the set-up of the private variables of ReporterIonAgent

Inspection Define the inspection logic that the Agent must perform.
The inspection of the Agent reports whether ITRAQ reporter ions appear at different intensities

Description Document the aim of this Agent.

These elements are illustrated by the DummyAgent in the following Java code snippet.

[12]The DummyAgent can be found in the Peptizer package be.proteomics.mat.util.agents. Click to browse the Java source code or the JavaDocs on this DummyAgent template to create custom Agents.

Algorithm 3 Agent signature in a code outline

```
public class DummyAgent extends Agent {

   //Parameters
   public static final String DUMMY_PROPERTY = "dummy";

   //Constructor
   public DummyAgent(){
      super();
      ...
   }

   //Inspection
   public AgentVote[] inspect(PeptideIdentification aPeptideIdentification){
      AgentVote vote = null;
      ...
      return vote;
   }

   //Description
   public String getDescription(){
      return "Agent description";
   }
}
```

Ok, so now being aware of the code signature of an Agent, the ReporterIonAgent can be created similarly.

PARAMETERS

First, the different **parameters** required by this Agent must be defined. Since these are read from the configuration file, they must have fixed identifiers. There are two parameters holding the values for the two **reporter ion masses** and a **fold ratio threshold** between the two reporter ion intensities for this Agent to inspect. Finally, there is also a parameter on the **error tolerance** that is allowed upon matching the reporter ion masses with a fragment ion from the MS/MS spectrum.

In Java, each of these is encoded as final static Strings, since these are then always **identical** and **accessible**.

The parameters for the ReporterIonAgent are illustrated in the following Java code snippet.

Algorithm 4 Parameters for the ReporterIonAgent

```
//Mass over charge value for the first reporter ion.
public static final String MASS_1 = "reporter_mz_1";

//Mass over charge value for the second reporter ion.
public static final String MASS_2 = "reporter_mz_2";

//Fold ratio between the two reporter ions.
public static final String RATIO = "ratio";

//Error tolerance for matching the expected
//reporter ion mass over charge to a fragment ion.
public static final String ERROR = "error";
```

Ok, after defining the parameters, the code for constructing the Agent can be written.

CONSTRUCTOR

The constructor is a special kind of routine as it is only once used upon starting a new Java object. The following parts can be recognized in the code:

Call superclass constructor to initiate all methods and variables common to all Agents at their superclass.

Read properties from the agent.xml configuration file for the Agent with this unique identifier.

Set general variables as given by the agent.xml configuration file to general agent variables like the name, the active and the veto status.

Set specific variables as given by the agent.xml configuration file to specific agent variables like the reporter ion masses, the ratio and the error tolerance.
Note that this is enclosed by a try & catch statement. If these variables are not inside the configuration file, then Peptizer will log an exceptional GUI message before shutting down the application.

The construction of the ReporterIonAgent is illustrated in the following Java code snippet:

Algorithm 5 Construction of an Agent

```
   /**
   * Construct a new instance of the ReporterIonAgent.
   */
   public ReporterIonAgent() {

      super();
      Properties prop = MatConfig.getInstance().getAgentProperties(this.getUniqueID());
      super.setName(prop.getProperty("name"));
      super.setActive(Boolean.valueOf(prop.getProperty("active")));
      super.setVeto(Boolean.valueOf(prop.getProperty("veto")));

      try {

         this.iProperties.put(MASS_1, prop.getProperty(MASS_1));
         this.iProperties.put(MASS_2, prop.getProperty(MASS_2));
         this.iProperties.put(RATIO, prop.getProperty(RATIO));
         this.iProperties.put(ERROR, prop.getProperty(ERROR));

      } catch (NullPointerException npe) {

         MatLogger.logExceptionalGUIMessage(
         "Missing Parameter!!", "Parameters " + MASS_1 + ", " + MASS_2 + " , " +
         RATIO + "and "+ ERROR + " are required! for Agent " + this.getName() +
         " !!\nExiting..");

         System.exit(0);
   }
}
```

With all this information, the ReporterIonAgent is ready to inspect an MS/MS spectrum for its reporter ions.

INSPECTION

The inspection is the core of an Agent since this logic leads to the Agent's vote. The **input** of the inspection is a PeptideIdentifcation object. Such an object has a single MS/MS spectrum and multiple peptide hypothesises. The **output** of the inspection is a vote as an AgentVote enumeration. There are three types of votes:

1. A vote approving to select the peptide hypothesis for a given property.
   Peptide hypothesises of MS/MS spectra with deviating reporter ion intensities

2. A vote being neutral to select the peptide hypothesis for a given property.
   Peptide hypothesises from MS/MS spectra with equal reporter ion intensities

3. A vote denying to select the peptide hypothesis for a given property.
Peptide hypothesises from MS/MS spectra lacking reporter ion fragment ions

Note that examples for the ReporterIonAgent shown in *italics* depend on expectations of the scenario, on what peptide hypothesises are interesting to the case that is initially defined. Therefore it is very important to document the Agents in depth.

This ReporterIonAgent is documented exhaustively so it is possible to read through to code step by step. As such, the source code of the ReporterIonAgent is included below. Comments are formatted in grey italics while Java keywords are blue and text values are green. The inspection part of the code starts from line 82. From thereon, the following parts can be recognized:

Preparing the variables (line 109) Here, a set of variables are defined that are needed to perform the inspection.
Variables to hold the observed peak intensity or matching status.

Inspecting for the reporter ions(line 145) Here, the actual inspection is performed.
Finding the reporter ions in the MS/MS spectrum, calculate their intensity ratio and test if it meets the expectations.

Making an inspection report and committing the votes(line 236) Here, the results of the inspection are stored and returned as an AgentVote for each peptide hypothesis.
Store the intensity ratio and the votes in a report that will be read by the AgentAggregator.

Fully documented source code for the ReporterIonAgent

```
1  package peptizer.agents.custom;
2
3  import be.proteomics.mascotdatfile.util.interfaces.Spectrum;
4  import be.proteomics.mascotdatfile.util.mascot.Peak;
5  import be.proteomics.mat.MatConfig;
6  import be.proteomics.mat.interfaces.Agent;
7  import be.proteomics.mat.util.AgentReport;
8  import be.proteomics.mat.util.PeptideIdentification;
9  import be.proteomics.mat.util.enumerator.AgentVote;
10 import be.proteomics.mat.util.fileio.MatLogger;
11
12 import java.math.BigDecimal;
13 import java.util.Properties;
14 /**
15  * Created by IntelliJ IDEA.
16  * User: kenny
17  * Date: 18-jun-2008
18  * Time: 15:30:28
19  */
20
21 /**
22  * Class description:
23  * ------------------
24  * This class was developed to inspect for deviating reporter ion intensities.
25  */
26 public class ReporterIonAgent extends Agent {
27
28     /**
29     * PARAMETERS
30     * ----------
31     * String identifiers for the parameters in the agent.xml configuration file.
32     */
33
34     // Mass over charge parameter for the first reporter ion.
35     public static final String MASS_1 = "reporter_mz_1";
36
37     // Mass over charge parameter for the second reporter ion.
38     public static final String MASS_2 = "reporter_mz_2";
39
40     // Fold ratio parameter between the two reporter ions you consider as deviating.
41     public static final String RATIO = "ratio";
42
43     // Error tolerance for matching the expected reporter
44     // ion mass over charge values to a fragmention in the MS/MS spectrum.
45     public static final String ERROR = "error";
46
47     /**
48     * CONSTRUCTOR
49     * -----------
50     * Construct a new instance of the ReporterIonAgent.
51     */
52     public ReporterIonAgent() {
53       // 1. Calls the constructor of the Agent superclass.
54       super();
55       // 2. Gets the properties for this Agent.
56       // A singleton MatConfig object reads the agent.xml configuration file upon starting Peptizer.
57       // The Properties of each Agent can be then be retrieved by the unique identifier of an Agent.
58       Properties prop = MatConfig.getInstance().getAgentProperties(this.getUniqueID());
59
60       try {
```

Fully documented source code for the ReporterIonAgent

```
61          // 2a. Sets common properties shared among all Agents.
62          super.setName(prop.getProperty("name"));
63          super.setActive(Boolean.valueOf(prop.getProperty("active")));
64          super.setVeto(Boolean.valueOf(prop.getProperty("veto")));
65
66          // 2b. Sets specific properties for this ReporterIonAgent.
67          // Masses of the two reporter ions, the threshold ratio and the mass error tolerance.
68          this.iProperties.put(MASS_1, prop.getProperty(MASS_1));
69          this.iProperties.put(MASS_2, prop.getProperty(MASS_2));
70          this.iProperties.put(RATIO, prop.getProperty(RATIO));
71          this.iProperties.put(ERROR, prop.getProperty(ERROR));
72      } catch (NullPointerException npe) {
73          // Note that an exception is thrown when one of the parameters is missing!
74          MatLogger.logExceptionalGUIMessage("Missing Parameter!!",
75                  "Parameters " + MASS_1 + ", " + MASS_2 + ", " + RATIO + "and " + ERROR +
76                          " are required! for Agent \"" + this.getName() + "\" !!\nExiting..");
77          System.exit(0);
78      }
79  }
80
81  /**
82  * INSPECTION
83  * ----------
84  * The inspection is the core of an Agent since this logic leads to the Agent's vote.
85
86  * This method returns an array of AgentVote objects, reflecting this Agent's idea
87  * whether to select or not to select the peptide hypothesis.
88
89  * All Agent Implementations must also create and store AgentReport for each peptide hypothesis.
90  *
91  * @param aPeptideIdentification PeptideIdentification that has to be inspected.
92  * @return AgentVote[] as a vote upon inspection for each the confident peptide hypothesises.
93  *       AgentVotes[0] gives the inspection result on PeptideHit 1
94  *       AgentVotes[1] gives the inspection result on PeptideHit 2
95  *       AgentVotes[n] gives the inspection result on PeptideHit n+1
96
97  *       Where the different AgentVotes can be:
98
99  *       a vote approving the selection of the peptide hypothesis.
100 *       a vote indifferent to the selection.
101 *       a vote objecting to select the peptide hypothesis.
102 */
103 public AgentVote[] inspect(PeptideIdentification aPeptideIdentification) {
104
105     // A. PREPARING THE VARIABLES
106     //****************************
107
108     // 1. The reporter ion masses.
109     double lReporterMass_1 = Double.parseDouble((String) (this.iProperties.get(MASS_1)));
110     double lReporterMass_2 = Double.parseDouble((String) (this.iProperties.get(MASS_2)));
111
112     // 2. The fold ratio threshold.
113     double lRatio = Double.parseDouble((String) (this.iProperties.get(RATIO)));
114
115     // 3. The error tolerance.
116     double lError = Double.parseDouble((String) (this.iProperties.get(ERROR)));
117
118     // 4. Reserves an array with AgentVotes for each confident peptide hypothesis.
119     AgentVote[] lAgentVotes =
120         new AgentVote[aPeptideIdentification.getNumberOfConfidentPeptideHits()];
```

Fully documented source code for the ReporterIonAgent

```
121
122         // Since this inspection is dependent on the MS/MS spectrum,
123         // it will result in the same vote for each peptide hypothesis.
124         // Therefore, a single inspection is reused for each peptide hypothesis.
125
126         // 5. Initiate an AgentReport serving as a report for this inspection.
127         iReport = new AgentReport(getUniqueID());
128
129         // 6. Local variable to store the result shown in the information table.
130         String lResultForTable = "";
131         // 7. Local variable to store the result written in the arff file.
132         String lResultForArff = "";
133
134         // 8. Local variables for matching reporter ion 1 in the MS/MS spectrum.
135         boolean lReporter_1_match = false;
136         double lReporter_1_intensity = 0;
137
138         // 9. Local variables for matching reporter ion 2 in the MS/MS spectrum.
139         boolean lReporter_2_match = false;
140         double lReporter_2_intensity = 0;
141
142         // B. THE ACTUAL INSPECTION
143         //**************************************
144
145         // 1. Gets the MS/MS spectrum from the PeptideIdentification object
146         // that was given as a parameter to the inspect() method.
147         Spectrum lSpectrum = aPeptideIdentification.getSpectrum();
148
149         // 2. Gets the peaklist from this MS/MS spectrum.
150         Peak[] lPeaks = lSpectrum.getPeakList();
151
152         // 3. Iterates over all peaks through a for loop.
153         for (int i = 0; i < lPeaks.length; i++) {
154             Peak lPeak = lPeaks[i];
155             double lDelta_1 = lPeak.getMZ() - lReporterMass_1;
156             double lDelta_2 = lPeak.getMZ() - lReporterMass_2;
157
158             // 3i) If absolute value of the mass diference of this fragmention and the expected mass
159             // of reporter ion 1 is less then the defined error tolerance.
160             // Then there is a match!
161             if (Math.abs(lDelta_1) < lError) {
162                 lReporter_1_match = true;
163                 lReporter_1_intensity = lPeak.getIntensity();
164             }
165
166             // 3ii) Same idea for reporter ion 2.
167             if (Math.abs(lDelta_2) < lError) {
168                 lReporter_2_match = true;
169                 lReporter_2_intensity = lPeak.getIntensity();
170             }
171
172             // 3iii) For performance reasons: if both peaks were matched then the for loop can be exited.
173             if (lReporter_1_match && lReporter_2_match) {
174                 break;
175             }
176         }
177
178         // 4. Checks the intensity ratio.
179         double lUpperBoundary;
180         double lLowerBoundary;
```

Fully documented source code for the ReporterIonAgent

```
181
182         // First an upper and a lower boundary must be defined for the Reporter Ions intensity ratio.
183         // If the experimental ratio between Reporter Ion 1 and Reporter Ion 2 is
184         //  more then 1.5 or less then 0.66, then the two samples deviate by a factor of 1.5.
185         //
186         // If the user defined the factor as larger then 1, the upper boundary for an
187         // deviating ratio is given by 1 diveded by that factor.
188         // The lower boundary for an deviating ratio is given by 1 multiplied by that factor.
189         // Example:
190         // If (Ratio=1.5)
191         // Then lower boundary = 1.5 and upper boundary = 0.66
192         if (lRatio > 1) {
193            lUpperBoundary = 1 * lRatio;
194            lLowerBoundary = 1 / lRatio;
195         } else {
196            // If the user defined the facotr as smaller then 1, then it is the other way round.
197            lUpperBoundary = 1 / lRatio;
198            lLowerBoundary = 1 * lRatio;
199         }
200
201         // 5. Local variable for the intesity ratio between Reporter Ion 1 and Reporter Ion 2.
202         double lExperimentalRatio;
203         // Local boolean for the upcomming function to store whether the ratio between the Reporter Ions
204         // deviates more then the expected ratio.
205         boolean lDeviatingRatio;
206
207         // 6. Checks the intensities of the reporter ions!
208
209         // 6i) If this condition is true, then one of the reporter ions was not found!
210         // These are not selected as these are probably unlabeled peptides.
211         if (!lReporter_1_match || !lReporter_2_match) {
212            lDeviatingRatio = false;
213            lExperimentalRatio = 0;
214            // 6ii) Else both the reporter ions were found. Lets inspect their ratio.
215         } else {
216            lExperimentalRatio = lReporter_1_intensity / lReporter_2_intensity;
217
218            // The ExperimentalRatio between reporter ion 1 an reporter ion 2
219            // is either less then the lower boundary,
220            // or either more then the upper boundar.
221            // In both cases, the reporter ion intesity is deviating for both samples:
222            if (lLowerBoundary > lExperimentalRatio || lUpperBoundary < lExperimentalRatio) {
223               // A. The Agent inspection resulted in deviating reporter ion intensities as
224               // their ratio was outside one of the boundaries.
225               lDeviatingRatio = true;
226            } else {
227               // B. Else the Agent inspection resulted in non deviating reporter ion intensities as
228               // their ratio was within the lower and upper boundary.
229               lDeviatingRatio = false;
230            }
231         }
232
233         // C. MAKING THE INSPECTION REPORTS AND COMMITTING THE VOTES
234         //*************************************************************
235
236         // 1. In all cases, store the experimental ratio between the
237         // two reporter ions as a value to display in the information table.
238
239         // A BigDecimal rounds a double at 2 decimals.
240         BigDecimal lRoundedExperimentalRatio = null;
```

Fully documented source code for the ReporterIonAgent

```
241         lRoundedExperimentalRatio = new BigDecimal(lExperimentalRatio).setScale(2, BigDecimal.ROUND
    _HALF_UP);
242         lResultForTable = lRoundedExperimentalRatio.toString();
243
244         // 2i. Deviating reporter ion intesity ratio, this Agent suggests to select the peptide hypothesis!
245         if (lDeviatingRatio) {
246           lResultForArff = "1";
247           for (int i = 0; i < lAgentVotes.length; i++) {
248             lAgentVotes[i] = AgentVote.POSITIVE_FOR_SELECTION;
249           }
250           // 2ii. Non Deviating reporter ion intensity ratio, this Agent is neutral to select the peptide hypothesis!
251         } else {
252           lResultForArff = "0";
253           for (int i = 0; i < lAgentVotes.length; i++) {
254             lAgentVotes[i] = AgentVote.NEUTRAL_FOR_SELECTION;
255           }
256         }
257
258         // 3. Creates an Agentreport for this inspection.
259         iReport.addReport(AgentReport.RK_RESULT, lAgentVotes[0]);
260         iReport.addReport(AgentReport.RK_TABLEDATA, lResultForTable);
261         iReport.addReport(AgentReport.RK_ARFF, lResultForArff);
262
263         // 4. Stores the report on the PeptideIdentification object.
264         for (int i = 0; i < lAgentVotes.length; i++) {
265           aPeptideIdentification.addAgentReport((i + 1), this.getUniqueID(), iReport);
266         }
267
268         // 5. Returns the AgentVotes in the end of the inspection.
269         return lAgentVotes;
270       }
271
272       /**
273       * Returns a description for the Agent.
274       * Note that html tags are used to stress properties.
275       * Use in tooltips and configuration settings.
276       * <p/>
277       * Fill in an agent description. Report on purpose and a minor on actual implementation.
278       *
279       * @return String description of the ReporterIonAgent.
280       */
281       public String getDescription() {
282         return "<html>Inspects for the abberant reporter ion intensities." +
283             "<b>Selects when two reporter ions ( " + this.iProperties.get(MASS_1) +
284             " , " + this.iProperties.get(MASS_2) + ") have a more then " + this.iProperties.get(RATIO) +
285             " fold intesity ratio.";
286       }
287     }
```

DESCRIPTION

The final method that must be implemented for each Agent is a descriptive method. Here resides the hard coded description the user reads in the Agent table upon starting a new Peptizer selection task. In addition, when the user is validating a Peptide hypothesis this description shows up in the information table.

It is important to stress shortly what it does, but also how an Agent casts a vote.

Algorithm 6 Agent description

```
/**
* Describe the ReporterIonAgent shortly.
*/
public String getDescription() {
return "<html>Inspects for the abberant reporter ion intensities." +
            "<b>Selects when two reporter ions ( " + this.iProperties.get(MASS_1) +
            " , " + this.iProperties.get(MASS_2) + ") have a more then " +this.iProperties.get(RATIO) +
            " fold intensity ratio.";
}
```

4.5 Using a custom Agent in Peptizer

The custom ReporterIonAgent is ready to be used in Peptizer. Therefore, the ReporterIonAgent must be added to the agent configuration file. This includes information on the classpath and classname as well as the Agent's parameters (see 3.5 for more information on the configuration files). This looks as following:

```
<agent>
<uniqueid>peptizer.agents.custom</uniqueid>
<property name="name">Reporter Ion Agent</property>
<property name="active">true</property>
<property name="veto">false</property>
<property name="reporter_mz_1">114.1</property>
<property name="reporter_mz_2">117.1</property>
<property name="ratio">1.5</property>
<property name="error">0.2</property>
</agent>
```

When a new selection task is started , the ReporterIonAgent is availlable to inspect peptide hypothesises as illustrated below.

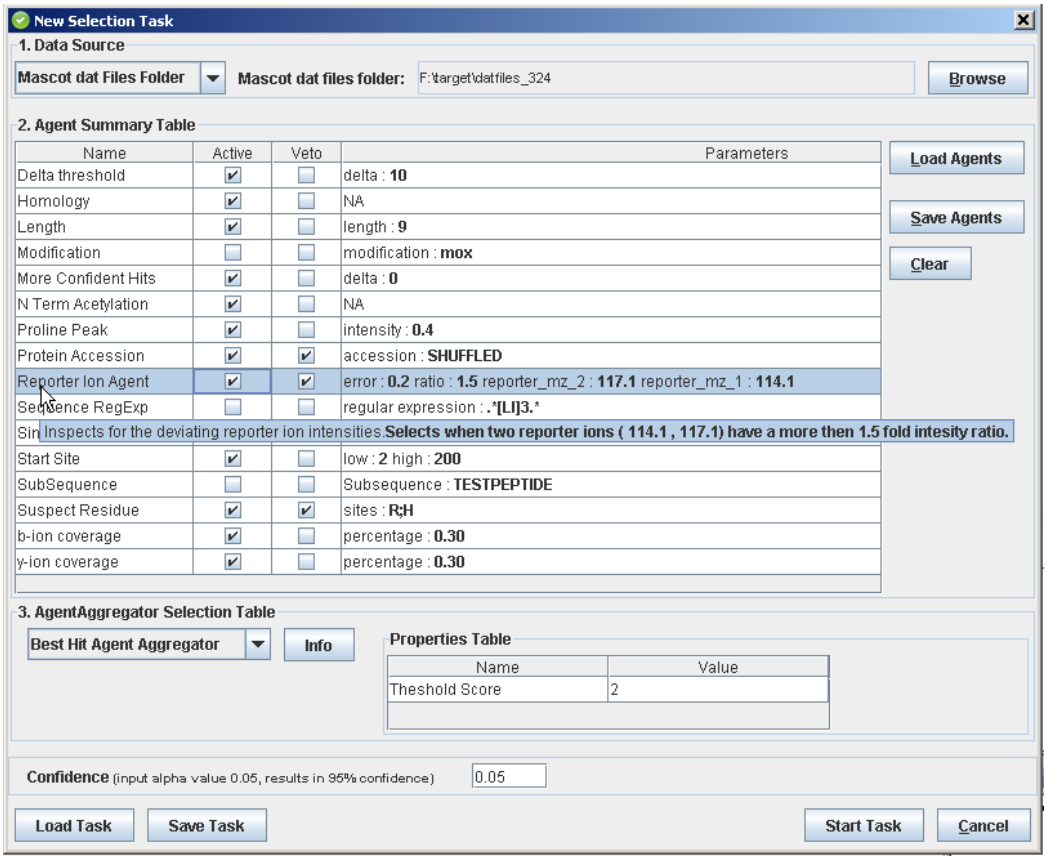


Figure 4.3: After adding the ReporterIon Agent to the agent configuration file, the ReporterIon Agent can be used for creating a new Peptizer task.

Peptizer will then inspect each MS/MS spectrum for deviating reporter ion intensities by using the ReporterIon Agent. As such, peptide hypothesises originating from MS/MS spectra with deviating reporter ion intensities will be selected and shown in the manual validation GUI of Peptizer. This is shown in the figure below. Note that this list can also be saved as a comma separated file (see 3.3).

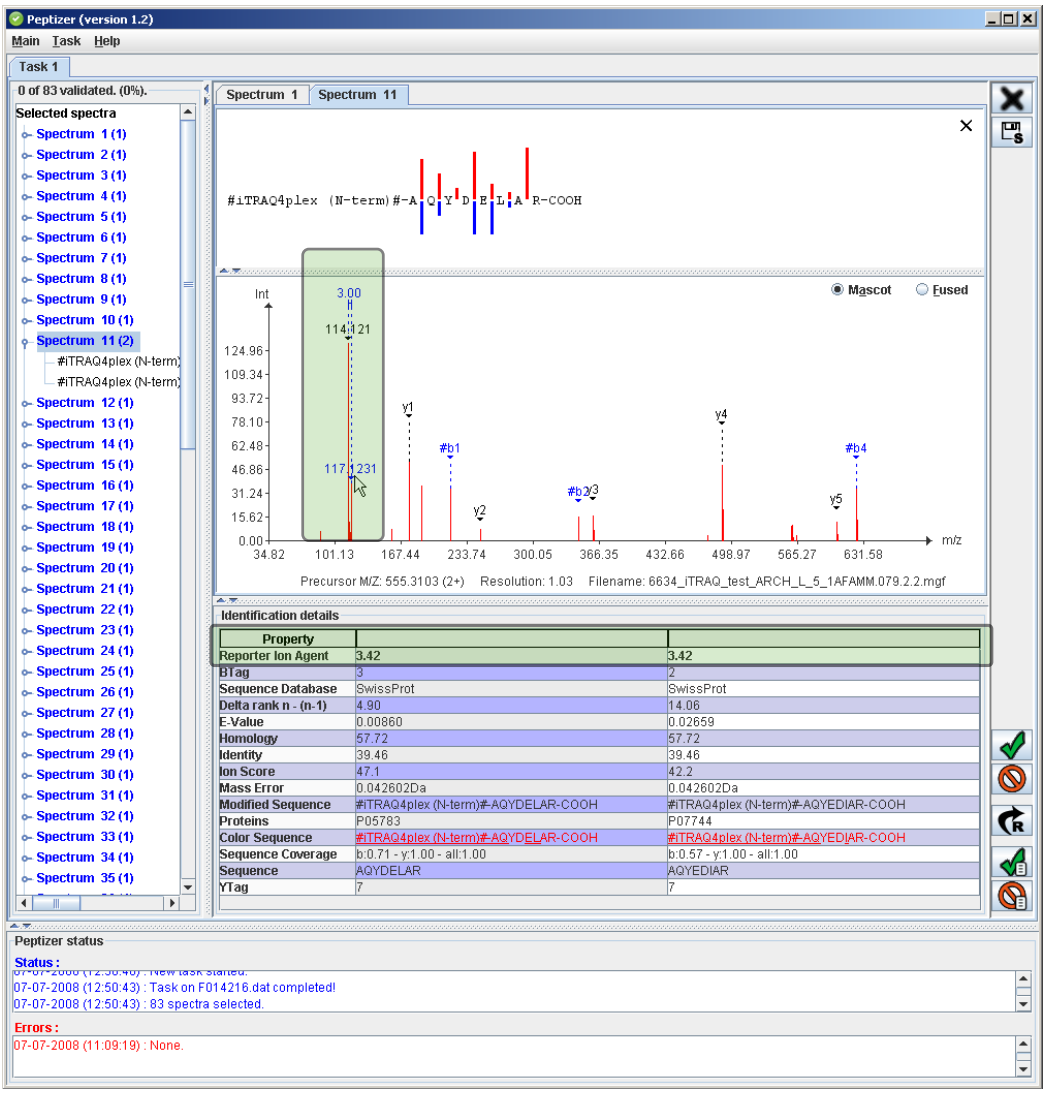


Figure 4.4: By using the ReporterIon Agent, Peptizer selected all peptide hypothesises from MS/MS spectra with deviating reporter ion intensities. Both green boxes show how the ReporterIon Agent first identifies the reporter ions in the MS/MS spectrum and used these to calculate the intensity ratio of **3.42**. Since this is more then the threshold ratio that was set to **1.5**, this peptide hypothesis was selected for its deviating reporter ion intesities in the MS/MS spectrum.

4.6 Writing your own AgentAggregator

Finally, when a series of Agents voted on a MS/MS spectrum and it's peptide hypothesises, these **Agent votes are input for an AgentAggregator**. The task of an AgentAggregator is then to bundle these votes and produce a conclusion whether or not the a PeptideIdentification matches a profile defined by its Agents.

The BestHitAggregator serves as an example for creating a custom AgentAggregator. The construction of a AgentAggregator is very similar to that of an Agent. First, all AgentAggregators must **extend the abstract AgentAggregator class** so to have a common set of variables and a **common signature**. Second, an AgentAggregator must also be **initialized** to set its properties from the aggregator.xml configuration file.

Algorithm 7 AgentAggregator construction

```
public class BestHitAggregator extends AgentAggregator {
    public static final String SCORE = "score";

    public BestHitAggregator() {
        initialize(SCORE);
    }
```

Also like in the Agent, the input/output structure of the AgentAggregator makes sense upon **implementing the abstract match method** from the AgentAggregator class. Again, a **PeptideIdentification object serves as an input parameter** and **a single AgentAggregatorResult is returned as output**. Note that the different aggregation results are static on the AgentAggregatorResult enum.

A collection of Agents is set upon starting a Peptizer task on the abstract AgentAggregator class. Therefore, an AgentAggregator implementation has no concern on the type of Agents, it must only be aware that there are some Agents ready for voting.

First, a number of local variables are declared that are used during the routine. Then, there is a check if there is any confident peptide hypothesis for this MS/MS spectrum. Only then starts **an iteration over all the availlable Agents**. Each Agent then **inspects the PeptideIdentification and returns an AgentVote**. As this is the BestHitAggregator, **only the votes for the best peptide hypothesis are taken into consideration**

here. During iteration, the veto status of an Agent is also logged, but only if the Agent is positive for selection.

Algorithm 8 BestHitAggregator matching method

```
public AgentAggregationResult match(PeptideIdentification aPeptideIdentification) {

    boolean boolConfident = false;
    boolean boolMatch = false;
    boolean boolVetoWasCalled = false;

    Integer lThresholdScore = new Integer(iProperties.getProperty(SCORE));

    int counter = -1;
    AgentVote[] results = new AgentVote[iAgentsCollection.size()];

    if (aPeptideIdentification.getNumberOfConfidentPeptideHits() > 0) {
        boolConfident = true;
        for (Agent lAgent : iAgentsCollection) {

            counter++;

            results[counter] = lAgent.inspect(aPeptideIdentification)[0];
            if (results[counter] == AgentVote.POSITIVE_FOR_SELECTION && lAgent.hasVeto()) {
                boolVetoWasCalled = true;
            }
        }

       if (boolVetoWasCalled) {
            boolMatch = true;
        } else {
            int lSumScore = sumVotes(results);

            if (lSumScore >= lThresholdScore) {
                boolMatch = true;
            }
        }

    }

    if (boolConfident) {
        if (boolMatch) {
            return AgentAggregationResult.MATCH;
        } else {
            return AgentAggregationResult.NON_MATCH;
        }
    } else {
        return AgentAggregationResult.NON_CONFIDENT;
    }
}
```

When the iteration has finished, a few lines of logic aggregate the votes. *First*, if an Agent with veto rights was positive for selection then it is a match for sure. *Second*, all votes are summed and compared with the scoring threshold as given by the user. If the sum is greater then the threshold, then it is a match. Else, a PeptideIdentification is not matched or not confident. The AgentAggregator concludes by returning a corresponding AgentAggregatorResult object.

In the end, the BestHitAggregator will thus have returned a conclusion on a given PeptideIdentification. Those PeptideIdentifications with a **AgentAggregatorResult.MATCH** result are subsequently presented in the **manual validation interface** of Peptizer.