OpenMS Tutorial

The OpenMS Developers

Mathias Walzer, Timo Sachsenberg, Fabian Aicheler,
George Rosenberger, Hannes Roest,
Marc Rurik, Stephan Aiche, Johannes Veit,
Knut Reinert, and Oliver Kohlbacher

Contents

1 General remarks

- This handout will guide you through an introductory tutorial for the OpenMS/TOPP software package [1].
- OpenMS [2] is a versatile open-source library for mass spectrometry data analysis. Based on this library, we offer a collection of command-line tools ready to be used by end users. These so-called TOPP tools (short for "The OpenMS Proteomics Pipeline") [3] can be understood as small building blocks of arbitrary complex data analysis workflows.
- In order to facilitate workflow construction, OpenMS was integrated into KNIME [4], the Konstanz Information Miner, an open-source integration platform providing a powerful and flexible workflow system combined with advanced data analytics, visualisation, and report capabilities. Raw MS data as well as the results of data processing using TOPP can be visualized using TOPPView [5].
- In this hands-on tutorial session, you will become familiar with some of the basic functionalities of OpenMS/TOPP, TOPPView, and KNIME and learn how to use a selection of TOPP tools used in the tutorial workflows.
- All data referenced in this tutorial can be found in the 🗀 `Example_Data` folder that came with this tutorial.

2 Getting started

2.1 Data conversion

Each MS instrument vendor has one or more formats for storing the acquired data. Converting these data into an open format (preferably mzML) is the very first step when you want to work with open-source mass spectrometry software. A freely available conversion tool is ProteoWizard. The OpenMS installation package for Windows automatically installs ProteoWizard, so you do not need to download and install it separately.

Please note that due to restrictions from the instrument vendors, file format conversion for most formats is only possible on Windows systems, so exporting from the acquisition PC connected to the instrument is usually the most convenient option. All files used in this tutorial have already been converted to mzML by us, so you do not need to do it yourself.

2.2 Data visualization using `TOPPView`

Visualizing the data is the first step in quality control, an essential tool in understanding the data, and of course an essential step in pipeline development. OpenMS provides a convenient viewer for some of the data: `TOPPView`.

We will guide you through some of the basic features of `TOPPView`. Please familiarize yourself with the key controls and visualization methods. We will make use of these later throughout the tutorial. Let's start with a first look at one of the files of our tutorial data set:

- Start `TOPPView` (see Start-Menu or Applications on MacOS)
- Go to File › Open File, navigate to the directory where you copied the contents of the USB stick to, and select 🗀 `OpenMS` ▸ `small` ▸ `velos005614.mzML` . This file contains a reduced LC-MS map (only a selected RT and m/z range was extracted using the TOPP tool FileFilter) of a label-free measurement of the human platelet proteome recorded on an Orbitrap velos. The other two mzML files contain technical replicates of this experiment.
- Play around.

- Three action modes are supported, one for translation, one for zooming and one for measuring:
  - Zoom mode
    - All previous zoom levels are stored in a zoom history. The zoom history can be traversed using ctrl + + or ctrl + - or the mouse wheel (scroll up and down).
    - Zooming into the data: either mark an area in the current view with your mouse while holding the left mouse button plus the ctrl key to zoom to this area or use your mouse wheel to traverse the zoom history.
    - If you have reached the end of the history and keep on pressing ctrl + + or scroll up, the current area will be enlarged by a factor of 1.25
    - Pressing the Backspace key resets the zoom and zoom history.
  - Translate mode
    - It is activated by default.
    - Move the mouse while holding the mouse button down to translate the current view (when zoomed in).
    - Arrow keys can be used to translate the view without entering translate mode.
  - Measure mode
    - It is activated using the ⇧ key.
    - Press the left mouse button down while a peak is selected and drag the mouse to another peak to measure the distance between peaks.
    - This mode is implemented in the 1D and 2D mode only.
- Right click on your 2D map and select Switch to 3D view and examine your data in 3D mode
- Go back to the 2D view. In 2D mode, visualize your data in different normalization modes, use linear, percentage and log-view (icons on the upper left tool bar).

> **Note:** On *Apple OS X*, due to a bug in one of the external libraries used by OpenMS, you will see a small window of the 3D mode when switching to 2D. Close the 3D tab in order to get rid of it.

- In `TOPPView` you can also execute TOPP tools. Go to Tools > Apply tool (whole layer) and choose a TOPP tool (e.g., FileInfo) and inspect the results.

2.3 Introduction to KNIME / OpenMS

Using OpenMS in combination with KNIME you can create, edit, open, save, and run workflows combining TOPP tools with the powerful data analysis capabilities of KNIME. Workflows can be created conveniently in a graphical user interface. The parameters of all involved tools can be edited within the application and are also saved as part of the workflow. Furthermore, KNIME interactively performs validity checks during the workflow editing process, in order to make it more difficult to create an invalid workflow.

Throughout most of the parts of this tutorial you will use KNIME to create and execute workflows. This first step is to make yourself familiar with KNIME.

2.3.1 Install OpenMS using KNIME

Before we can start with the tutorial we need to install all the required extension for KNIME. First we will install the OpenMS plugin, providing all the OpenMS nodes. Afterwards we will install some additional plugins that we will use in the more advanced part of this tutorial.

1. Open KNIME.
2. Click on Help > Install New Software...
3. In the now open dialog choose Add... (in the upper right corner of the dialog) to define a new update site. In the opening dialog enter the following details.
   Name: `Trunk Community Contributions`
   Location: `http://tech.knime.org/update/community-contributions/trunk/`
4. After pressing OK KNIME will show you all the contents of the added Update Site, containing also the OpenMS nodes.

5. Select the OpenMS nodes in the category ”KNIME Community Contributions - Bioinformatics & NGS” and click Next.
6. Follow the instructions and after a restart of KNIME the OpenMS nodes will be available under “Community Nodes”.

> **Note:** For this tutorial we use a pre-release version of OpenMS 1.12. While not being a full release, it was nevertheless intensively tested to ensure its functionality for this tutorial. For regular use we recommend using the latest stable OpenMS release. To install the latest stable release skip Steps 3 and 4 and instead choose the update site Trusted Community Contributions. Please note that some of the workflows shown here require OpenMS 1.12 and therefore will not work with OpenMS 1.11.1 downloaded from the stable update site.

For the rest of the tutorial we will also need some more plugins, that can be installed similar to the OpenMS nodes.

1. Again, click on Help > Install New Software...
2. From the Work with: drop down list select the KNIME Analytics Platform Update Site
3. Now select the following plugins from the *KNIME & Extensions* category
   - KNIME Base Chemistry Types & Nodes
   - KNIME Chemistry Add-Ons
   - KNIME File Handling Nodes
   - KNIME Interactive R Statistics Integration
   - KNIME Math Expression (JEP)
   - KNIME Report Designer
   - KNIME SVG Support
   - KNIME XLS Support
   - KNIME XML-Processing

4. And the following plugin from the *Marvin Chemistry Extensions (donated by Infocom & Chemaxon)* category
    - ChemAxon/Infocom Marvin Extensions Feature
5. Click Next and follow the instructions.

2.3.2 KNIME Concepts

A **workflow** is a sequence of steps applied to a single or multiple input data sets to process and analyse this data. In KNIME such workflows are implemented graphically by combining so-called **nodes**.

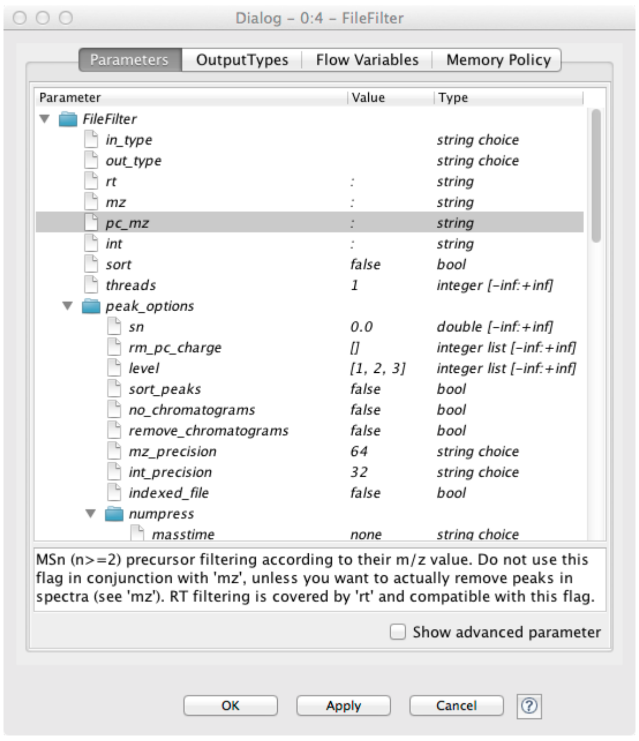
A node represents a single analysis step in a workflow. Nodes have input and output ports where the data goes into to the node or the results are provided for other nodes after processing, respectively. KNIME distinguishes between different port types, representing different types of data. The most common representation in KNIME are tables (similar to an excel sheet). Those ports are marked with a small triangle. For OpenMS we use a different port type, so called file ports, representing complete files. Those ports are marked by a small grey box. Dark grey boxes represent mandatory inputs and light grey boxes optional inputs.

Nodes can have three different states, indicated by the small traffic light below the node.

- Inactive, failed, and not yet fully configured nodes are marked red.
- Configured but not yet executed nodes are marked yellow.
- Successfully executed nodes are marked green.

If the node execution failed the node will switch to the red state.

Most nodes will be configured as soon as all input ports are connected. For some nodes additional parameters have to be provided that cannot be either guessed from the data or filled with sensible defaults. In this case, of if you want to customise the default configuration, you can open the configuration dialog of a node with a double-click on the node. For OpenMS you will see a configuration dialog like the one shown in Figure 1.



Parameter	Value	Type
FileFilter		
in_type		string choice
out_type		string choice
rt	:	string
mz	:	string
pc_mz	:	string
int	:	string
sort	false	bool
threads	1	integer [-inf:+inf]
peak_options		
sn	0.0	double [-inf:+inf]
rm_pc_charge	[]	integer list [-inf:+inf]
level	[1, 2, 3]	integer list [-inf:+inf]
sort_peaks	false	bool
no_chromatograms	false	bool
remove_chromatograms	false	bool
mz_precision	64	string choice
int_precision	32	string choice
indexed_file	false	bool
numpress		
masstime	none	string choice

MSn (n>=2) precursor filtering according to their m/z value. Do not use this flag in conjunction with 'mz', unless you want to actually remove peaks in spectra (see 'mz'). RT filtering is covered by 'rt' and compatible with this flag.

☐ Show advanced parameter

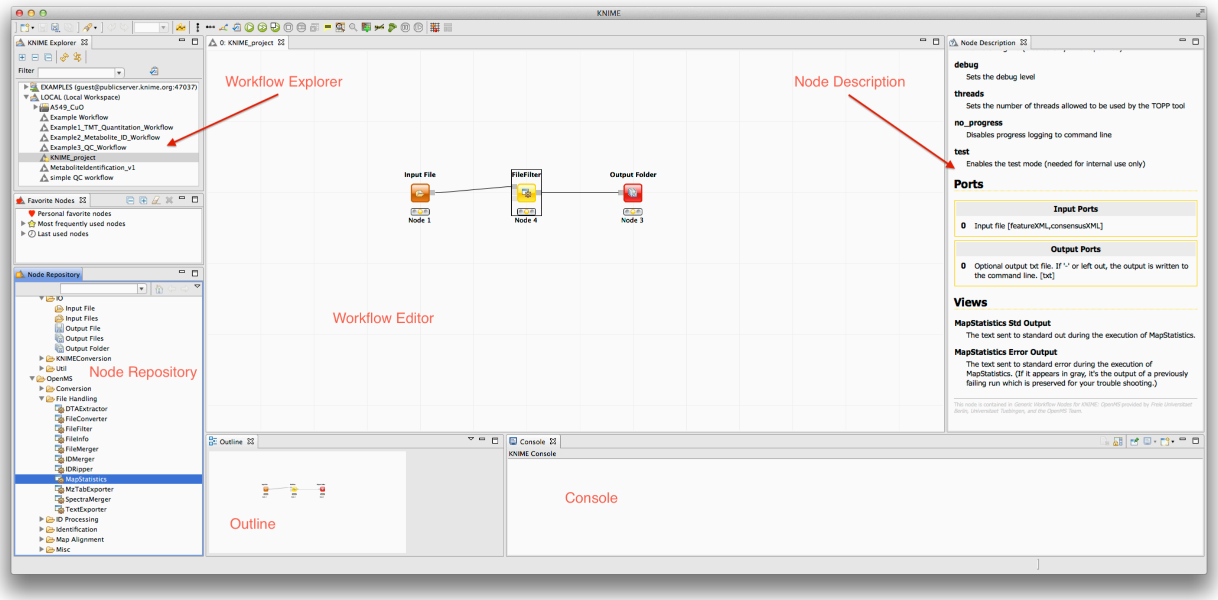
Figure 1: Node configuration dialog of an OpenMS node.

Note: OpenMS distinguishes between normal parameters and advanced parameters. Advanced parameters are by default hidden from the users since they should only rarely be customised. In case you want to have a look at the parameters or need to customise them in one of the tutorials you can show them by clicking on the checkbox `Show advanced parameter` in the lower part of the dialog.

The dialog shows the individual parameters, their current value and type, and, in the lower part of the dialog, the documentation for the currently selected parameter.

2.3.3 Overview of the graphical user interface

The graphical user interface (GUI) of KNIME consists of different components or so called panels that are shown in Figure 2. We will shortly introduce the individual panels and their purpose below.



debug
Sets the debug level

threads
Sets the number of threads allowed to be used by the TOPP tool

no_progress
Disables progress logging to command line

test
Enables the test mode (needed for internal use only)

Ports

Input Ports	
0	Input file [featureXML,consensusXML]

Output Ports	
0	Optional output txt file. If '-' or left out, the output is written to the command line. [txt]

Views

MapStatistics Std Output
The text sent to standard out during the execution of MapStatistics.

MapStatistics Error Output
The text sent to standard error during the execution of MapStatistics. (If it appears in gray, it's the output of a previously failing run which is preserved for your trouble shooting.)

This node is contained in *Generic Workflow Nodes for KNIME: OpenMS* provided by *Freie Universitaet Berlin, Universitaet Tuebingen, and the OpenMS Team.*

Figure 2: The KNIME workbench.

Workflow Editor The workflow editor is the central part of the KNIME GUI. Here you assemble the workflow by adding nodes from the Node Repository via "drag & drop". Nodes can be connected by clicking on the output port of one node and releasing the mouse at the desired input port of the next node.

Workflow Explorer Shows a list of available workflows (also called workflow projects). You can open a workflow by double clicking it. A new workflow can be created with a right-click in the Workflow Explorer followed by selecting New KNIME Workflow... .

Node Repository Shows all nodes that are available in your KNIME installation. Every plugin you install will provide new nodes that can be found here. The OpenMS nodes can be found in Community Nodes > OpenMS. Nodes for managing files (e.g., Input Files or Output Folders) can be found in Community Nodes > GenericKnimeNodes. You can search the node repository by typing the node name into the small text box in the upper part of the node repository.

Outline The Outline panel contains a small overview of the complete workflow. While of limited use when working on a small workflow, this feature is very helpful as soon as the workflows get bigger.

Console In the console panel warning and error messages are shown. This panel will provide helpful information if one of the nodes failed or shows an warnings sign.

Node Description As soon as a node is selected, the Node Description window will show the documentation of the node including documentation for all its parameters. For OpenMS nodes you will also find a link to the tool page in the online documentation.

2.3.4 Creating workflows

Workflows can easily be created by a right click in the Workflow Explorer followed by clicking on New KNIME Workflow... .

2.3.5 Sharing workflows

To be able to share a workflow with others KNIME supports the import and export of complete workflows. To export a workflow select it in the Workflow Explorer and select File > Export KNIME Workflow... . KNIME will export workflows as a zip file containing all the information on nodes, their connections, and their configuration. Those zip files can again be imported by selecting File > Import KNIME Workflow... .

> **Note:** For your convenience we added all workflows discussed in this tutorial to the 🗀 `Workflows` folder. If you want to check your own workflow by comparing it to the solution or got stuck, simply import the full workflow from the corresponding zip file.

2.3.6 Duplicating workflows

During the tutorial a lot of the workflows will be created based on the workflow from a previous task. To keep the intermediate workflows we suggest you create copies of your workflows so you can see the progress. To create a copy of your workflow follow the following steps

- Right click on the workflow you want to create a copy of in the Workflow Explorer and select Copy .

- Right click again somewhere on the workflow explorer and select Paste.
- This will create a workflow with same name as the one you copied with a (2) appended.
- To distinguish them later on you can easily rename the workflows in the Workflow Explorer by right clicking on the workflow and selecting Rename.

  > **Note:** To rename a workflow it has to be closed.

2.3.7 A minimal workflow

Let us now start with the creation of our very first, very simple workflow. As a first step, we will gather some basic information about the data set before starting the actual development of a data analysis workflow.

- Create a new workflow.
- Add an `Input File` node and an `Output Folder` node (to be found in Community Nodes > GenericKnimeNodes > IO and a `FileInfo` node (to be found in the category Community Nodes > OpenMS > File Handling) to the workflow.
- Connect the `Input File` node to the `FileInfo` node, and the first output port of the `FileInfo` node to the `Output Folder` node.

  > **Note:** In case you are unsure about which node port to use, hovering the cursor over the port in question will display the port name and what kind of input it expects.

  The complete workflow is shown in Figure 3. FileInfo can produce two different kinds of output files.
- All nodes are still marked red, since we are missing an actual input file. Double-click the Input File node and select Browse. In the file system browser select 🗀 `OpenMS` ▸ `tiny` ▸ `velos005614.mzML` and click Open. Afterwards close the dialog by clicking Ok.

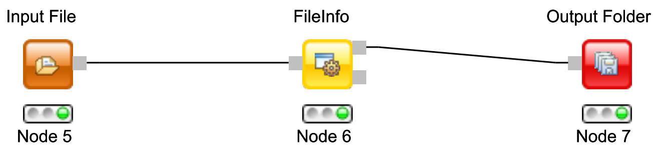


Figure 3: A minimal workflow calling FileInfo on a single file.

> **Note:** Make sure to use the "tiny" version this time, not "small", for the sake of faster workflow execution.

- The `Input File` node and the `FileInfo` node should now have switched to yellow, but the `Output Folder` node is still red. Double-click on the `Output Folder` node and click on Browse to select an output directory for the generated data.
- Great! Your first workflow is now ready to be run. Press ⇧ + F7 to execute the complete workflow. You can also right click on any node of your workflow and select Execute from the context menu.
- The traffic lights tell you about the current status of all nodes in your workflow. Currently running tools show either a progress in percent or a moving blue bar, nodes waiting for data show the small word "queued", and successfully executed ones become green. If something goes wrong (e.g., a tool crashes), the light will become red.
- In order to inspect the results, you can just right-click the `Output Folder` node and select View: Open the output folder. You can then open the text file and inspect its contents. You will find some basic information of the data contained in the mzML file, e.g., the total number of spectra and peaks, the RT and m/z range, and how many MS1 and MS2 spectra the file contains.

Now consider you would like to gather this information for more then one file. We will now modify the workflow to compute the same information on three different files and then write the output files to a folder.

- We start from the previous workflow.

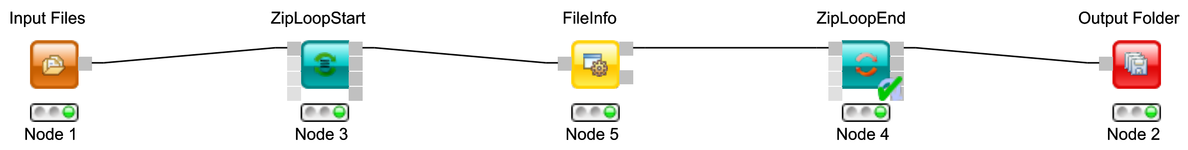


Figure 4: A minimal workflow calling FileInfo on multiple files in a loop.

- First we need to replace our single input file with multiple files. Therefore we add the `Input Files` node from the category Community Nodes › GenericKnimeNodes › IO.
- To select the files we double-click on the `Input Files` node and click on Add. In the filesystem browser we select all three files from the directory `OpenMS` ▸ `tiny`. And close the dialog with Ok.
- We now add two more nodes: the `ZipLoopStart` and the `ZipLoopEnd` node from the category Community Nodes › GenericKnimeNodes › Flow.
- Afterwards we connect the `Input Files` node to the first port of the `ZipLoopStart` node, the first port of the `ZipLoopStart` node to the `FileInfo` node, the first output port of the `FileInfo` node to the first input port of the `ZipLoopEnd` node, and the first output port of the `ZipLoopEnd` node to the `Output Folder` node. The complete workflow is shown in Figure 4
- The workflow is already complete. Simply execute the workflow and inspect the output as before.

3 Label-free quantification

3.1 Introduction

In this chapter, we will build a workflow with OpenMS / KNIME to quantify a label-free experiment. Label-free quantification is a method aiming to compare the relative amounts of proteins or peptides in two or more samples. We will start from the minimal workflow from the last chapter and, step-by-step, build a label-free quantitation workflow.

3.2 Peptide Identification

As a start, we will extend the minimal workflow so that it performs a peptide identification using the OMSSA [6] search engine. Since OpenMS version 1.10, OMSSA is included in the OpenMS installation, so you do not need to download and install it yourself.

- Instead of FileInfo, we want to perform OMSSA identification, so we simply replace the `FileInfo` node with the `OMSSAAdapter` node Community Nodes › OpenMS › Identification, and we are almost done. Just make sure you have connected the `ZipLoopStart` node with the `in` port of the `OMSSAAdapter` node.
- OMSSA, like most mass spectrometry identification engines, relies on searching the input spectra against sequence databases. Thus, we need to introduce a search database input. As we want to use the same search database for all of our input files, we can just add a single `Input File` node to the workflow and connect it directly with the `OMSSAAdapter database` port. KNIME will automatically reuse this Input node each time a new ZipLoop iteration is started. In order to specify the database, select 🗀 `OpenMS▸FASTA▸fastafile▸uniprot_sprot_101104_human_concat.fasta`, and we have a very basic peptide identification workflow.

  > **Note:** We recommend to choose a different output directory every time you extend and run your pipeline again.

  > **Note:** You might also want to save your new identification workflow under

> a different name. Have a look at Section 2.3.6 for information on how to create copies of workflows.

- The result of a single OMSSA run is basically a number of peptide-spectrum-matches (PSM) with a score each, and these will be stored in an idXML file. Now we can run the pipeline and after execution is finished, we can have a first look at the results: just open the input files folder with a file browser and from there open a mzML file in `TOPPView`.
- Here, you can annotate this spectra data file with the peptide identification results. Choose Tools » Annotate with identification from the menu and select the idXML file that `OMSSAAdapter` generated (it is located within the output directory that you specified when starting the pipeline).
- On the right, select the tab Identification view. Using this view, you can see all identified peptides and browse the corresponding MS2 spectra.

  > **Note:** Opening the output file of `OMSSAAdapter` (the idXML file) directly is also possible, but the direct visualization of an idXML file is less useful.

As you can see, the spectra are annotated with an unusually high number of identifications, many of which are probably false positives. Therefore, we will tweak the parameters of OMSSA to better reflect the instruments accuracy of our measurements. Also, we will extend our pipeline with a false discovery rate (FDR) filter to retain only those identifications that will yield an FDR of < 1 %.

- Open the configuration dialog of `OMSSAAdapter`. Since we know that data was acquired using an Orbitrap velos instrument, we can set the precursor mass tolerance to a smaller value, say 10 ppm. Set *precursor_ mass_ tolerance* to 10 and *precursor_mass_tolerance_unit_ppm* to *true*.

  > **Note:** Whenever you change the configuration of a node the node as well as all its successors will be reset to the Configured state.

- Set *max_precursor_charge* to 4, in order to also search for peptides with charges up to 4.

- Add Carbamidomethyl (C) as fixed modification and Oxidation (M) as variable modification.

  > **Note:** To add a modification click on the empty value field in the configuration dialog to open the list editor dialog. In the new dialog click Add. Then select the newly added modification (15dB-biotin (C)) to open the drop down list where you can select the correct modification.

- A common step in analyis is to search not only against a regular protein database, but to also search against a decoy database for FDR estimation. The fasta file we used before already contains such a decoy database. For OpenMS to know which OMSSA PSM came from which part of the file (i.e. regular versus decoy), we have to index the results. Therefore extend the workflow with a `PeptideIndexer` node Community Nodes > OpenMS > ID Processing. This node needs the idXML as input as well as the database file.

  > **Note:** You can direct the files of an `Input File` node to more than just one destination port.

- The decoys in the database are prefixed with “sw”, so we have to set *decoy_string* to sw and *prefix* to true in the configuration dialog of `PeptideIndexer` accordingly.

- Now we can go for the FDR estimation, which will the `FalseDiscoveryRate` node calculate for us Community Nodes > OpenMS > ID Processing. As we have a combined search database and thus only one idXML per mzML we will only use the in port of the `FalseDiscoveryRate` node.

- In order to set the FDR level to 1%, we need an `IDFilter` node from Community Nodes > OpenMS > ID Processing. Configuring its parameter *score* $\rightarrow$ *pep* to 0.01 will do the trick. The FDR calculations (embedded in the idXML) from the `FalseDiscoveryRate` node will go into the in port of the `IDFilter` node.

- Execute your workflow and inspect the results using the identification view like you did before. You can export the list of identified peptides using the export table button. How many peptides did you identify at this FDR threshold?

Note: The finished identification workflow is now sufficiently complex that we might want to encapsulate it in a Meta node. For this, select all nodes inside the ZipLoop (excluding the Input File node) and right-click to select Collapse into Meta node and name it ID. Meta nodes are useful when you construct even larger workflows and want to keep an overview.

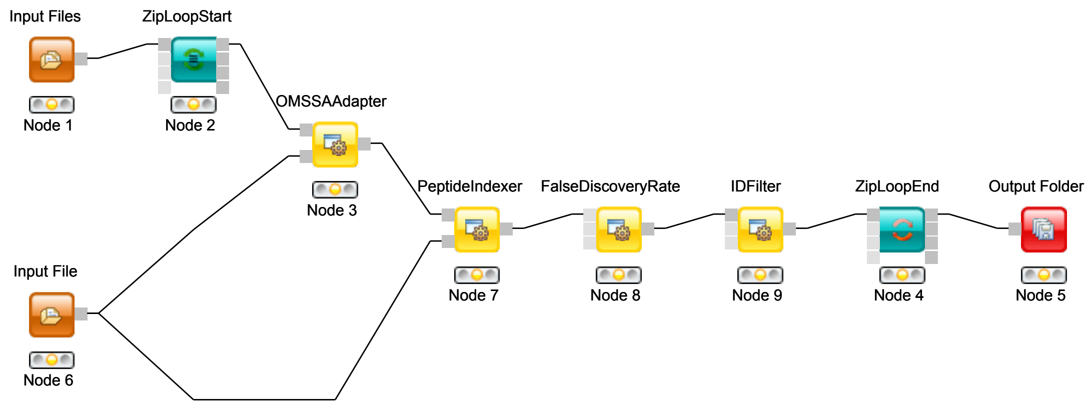


Figure 5: OMSSA ID pipeline including FDR filtering

3.2.1 Bonus task: identification using several search engines

Note: If you are ahead of the tutorial or later on, you can further improve your FDR identification workflow by a so-called consensus identification using several search engines.

It has become widely accepted that the parallel usage of different search engines can increase peptide identification rates in shotgun proteomics experiments. The ConsensusID algorithm is based on the calculation of posterior error probabilities (PEP) and a combination of the normalized scores by considering missing peptide sequences.

- Next to the `OMSSAAdapter` add a `XTandemAdapter` Community Nodes > OpenMS > Identification node and set its parameters and ports analogously to the `OMSSAAdapter`.
- To calculate the PEP, introduce each a `IDPosteriorErrorProbability` Community Nodes > OpenMS > ID Processing node to the output of each ID engine adapter node. This will calculate the PEP to each hit and output an updated idXML.

- To create a consensus we must first merge these two files with a `FileMerger` node Community Nodes > GenericKnimeNodes > Flow so we can then merge the corresponding IDs with a `IDMerger` Community Nodes > OpenMS > File Handling.
- Now we can create a consensus identification with the `ConsensusID` Community Nodes > OpenMS > ID Processing node. We can connect this to the `PeptideIndexer` and go along with our existing FDR filtering.

> **Note:** By default, X!Tandem takes additional enzyme cutting rules into consideration (besides the specified tryptic digest). Thus you have to set PeptideIndexer's *enzyme* → *specificity* parameter to `semi` to accept X!Tandems semi tryptic identifications as well.

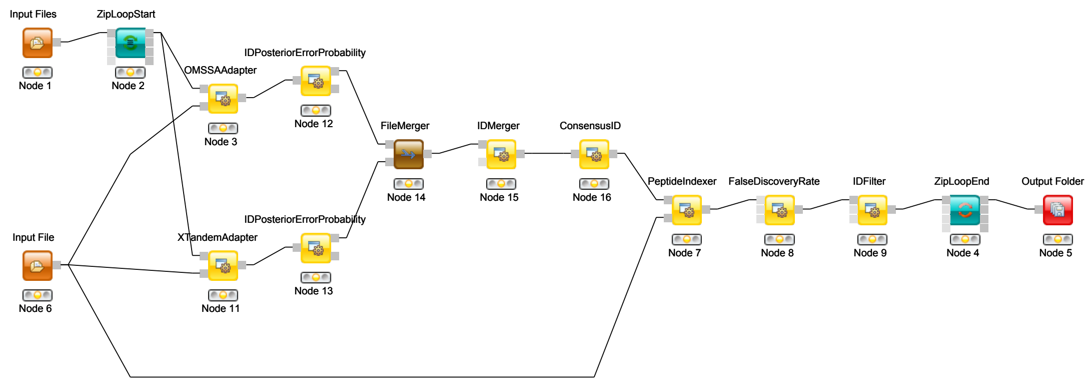


Figure 6: Complete consensus identification workflow

3.3 Quantification

Now that we have successfully constructed a peptide identification pipeline, we can add quantification capabilities to our workflow.

- Add a `FeatureFinderCentroided` node Community Nodes > OpenMS > Quantitation which gets input from the first output port of the `ZipLoopStart` node. Also, add an `IDMapper` node Community Nodes > OpenMS > ID Processing which gets input from the `FeatureFinderCentroided` node and the ID Meta node (or `IDFilter` node if you haven't used the Meta node). The output of the `IDMapper` is then connected to the `ZipLoopEnd` node.

- `FeatureFinderCentroided` finds and quantifies peptide ion signals contained in the MS1 data. It reduces the entire signal, i.e., all peaks explained by one and the same peptide ion signal, to a single peak at the maximum of the chromatographic elution profile of the monoisotopic mass trace of this peptide ion and assigns an intensity corresponding to the area under the monoisotopic mass trace.

- `FeatureFinderCentroided` produces a featureXML file as ouput, containing only quantitative information of so-far unidentified peptide signals. In order to annotate these with the corresponding ID information, we need the `IDMapper` node.

- Run your pipeline and inspect the results of the `IDMapper` node in TOPPView.

- In order to assess how well the feature finding worked, you can project the features contained in the featureXML file on the raw data contained in the mzML file. In TOPPView choose File › Open file and select `OpenMS ▸ tiny ▸ velos005614.mzML`. In the dialog that pops up, select Open in › New layer. Zoom in until you see boxes (found features) around the peptide signals in the raw data.

  > **Note:** The RT range is very narrow. Thus, select the full RT range and zoom only into the m/z dimension by holding down CTRL (CMD on the Mac) and repeatedly dragging a narrow box from the very left to the very right.

- You can see which features were annotated with a peptide identification by first selecting the featureXML file in the `Layers` window on the upper right side and then clicking on the icon with the letters A, B and C on the upper icon bar. Now, click on the small triangle next to that icon and select `Peptide identification`.

3.4 Combining quantitative information across several label-free experiments

So far, we successfully performed peptide identification as well as quantification on individual LC-MS maps. For differential label-free analyses however, we need to identify and

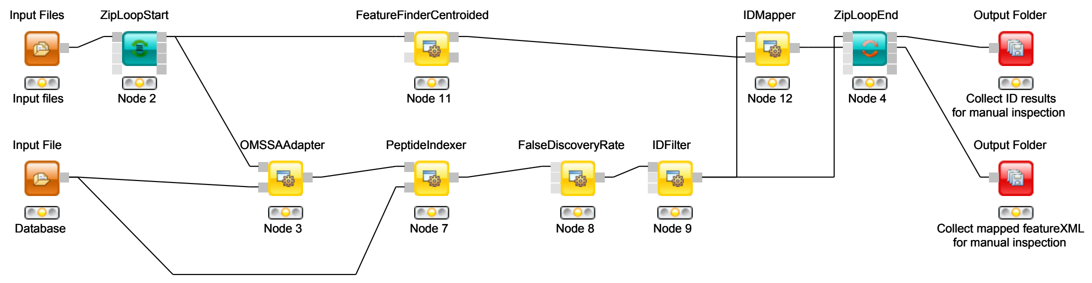


Figure 7: Extended workflow featuring peptide identification and quantification

quantify corresponding signals in different experiments and link them together to compare their intensities. Thus, we will now run our pipeline on all three available input files (replicates) and extend it a bit further, so that it is able to find and link features across several maps.

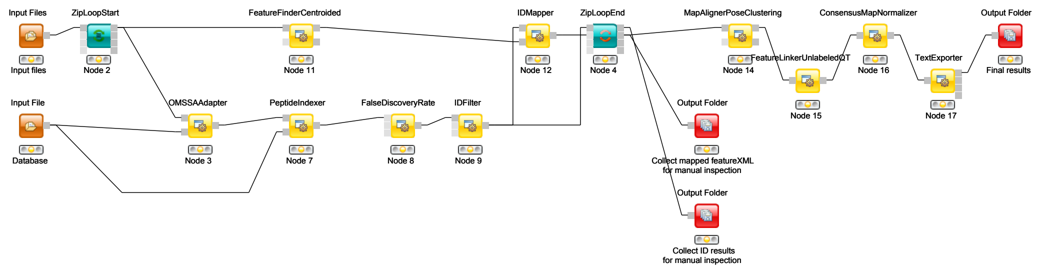


Figure 8: Complete identification and label-free quantification workflow

- To find features across several maps, we first have to align them to correct for retention time shifts between the different label-free measurements. With the `MapAlignerPoseClustering` Community Nodes > OpenMS > Map Alignment, we can align corresponding peptide signals to each other as closely as possible by applying a transformation in the RT dimension.

  > **Note:** `MapAlignerPoseClustering` consumes several featureXML files and its output should still be several featureXML files containing the same features, but with the transformed RT values. In its configuration dialog, make sure the OutputTypes is set to featureXML.

- With the `FeatureLinkerUnlabeledQT` node Community Nodes > OpenMS > Map Alignment, we can then perform the actual linking of corresponding features. Its output is a consensusXML file containing linked groups of corresponding features across the different experiments.

- Since the overall intensities can vary a lot between different measurements (for example, because the amount of injected analytes was different), we apply the `ConsensusMapNormalizer` Community Nodes » OpenMS » Map Alignment as a last processing step. Configure its parameters with setting *algorithm_type* to `median`. It will then normalize the maps in such a way that the median intensity of all input maps is equal.

- Finally, we export the resulting normalized consensusXML file to a csv format using `TextExporter` Community Nodes » OpenMS » File Handling. Connect its out port to a new `Output Folder` node.

> **Note:** You can specify the desired column separation character in the parameter settings (by default, it is set to “ ” (a space)). The output file of `TextExporter` can then be opened with external tools, e.g., Microsoft Excel, for downstream statistical analyses.

3.4.1 Bonus task: data analysis in KNIME

For downstream analysis of the quantification results, you can use the `ConsensusTextReader` node instead of the `Output Folder` node to convert the output into a KNIME table (indicated by an arrow as output port). After running the node you can view the KNIME Table by right clicking on the `ConsensusTextReader` selecting Consensus Table. The output of the node should now be compatible with most of the nodes of the KNIME base as well as the R nodes, which leaves room for you to play with these. Possible analyses include:

Task

✓ Filtering (`Row Filter`, `Rule-based Row Filter`) or grouping (`GroupBy`; e.g. by charge) of the identified peptides/proteins.

Task

Statistical analysis with `R Snippet` nodes or the nodes from the KNIME Statistics package.

Task

Plotting of the (cumulative) distribution of q-values, quality scores (of the consensus features) or other peptide hit properties using `R View` nodes or the standard KNIME nodes for plotting (which include interactive functionality).

Task

For further inspiration you might take a look at Chapter 5.5 which is describing different types of analyses using KNIME on metabolomics data.

4 Quality Control

4.1 Introduction

Quality **C**ontrol is an important part of mass spectrometry experiments and analyses. However, workflows and quality control protocols may differ vastly between labs. For this reason, quality control in OpenMS is designed to be very flexible, using a userdefined set of metrics from a controlled vocabulary (CV), the QC-CV. By using an own CV, Quality Reports are not limited in its form. Thus, results can be customized and still fit a standard quality report template. Reports are easy to evaluate due to the design of the used format, qcML.

4.2 QC-CV

A controlled vocabulary is a list of entries defining each a phrase or keyword in a given context. In our case, a controlled vocabulary entry of a metric can describe both experimental as well as programmatic environmental variables. It is comprised of a name, an identifier, and a definition. This definition describes the metric and what aspect of quality control is conducted with such a metric.
A CV entry also has associated relations, like *Chromatogram count* is a *MS aquisition result details*. This gives the CV a hierarchical structure and should make it easier to comprehend and also easier to browse (`http://www.ebi.ac.uk/ontology-lookup/browse.do?ontName=qcML`).
Like this, a set of quality metrics can be chosen according to the requirement of the lab, researcher or workflow. The content of the vocabulary is community defined. If your favourite metric, with which you control the quality of your experiments, does not yet have an entry, feel urged to propose its addition.
Inside a qcML file, you will have a bunch of quality parameters. These belong to a specific MS run (or set) and combine a value with a CV entry., e.g. 2069 with *MS2 spectra count*.

4.3 Single run QC

We will start by adapting the label-free quantification workflow as it comprises all the basic analyses we need to begin with:

- The spectra data itself,
- the identifications to the spectra and
- the features found on which quantification is based.

Calculating basic statistics and agglomerating quality data will be done by the `QCCalculator` node from Community Nodes › OpenMS › Utilities. It will also be needed for more advanced quality metrics later on. The QCCalculator node consumes MS runs (mzML), identifications (idXML), and found features (featureXML). The output are quality parameters which are stored in a qcML file. This will make it easy to access the quality data we need. The qcML file will be handed down from one metric calculation or visualization to another. Each QC step will append its plot, table, or value. With the qcML file you can also instantly visualize the metrics you have calculated so far: qcML includes all information, renderable in all recent browsers, even without an internet connection.
At this point, the qcML will not contain much to look at, so we will start filling it - with a heatmap-like (RT over M/Z, Intensity color coded) plot of the mass spectrometry experiment itself. We can extract that from the mzML with an ImageCreator node from Community Nodes › OpenMS › Utilities. The plot can then be integrated to the qcML with the QCEmbedder node from Community Nodes › OpenMS › Utilities. This node is capable to embed either images or tables into the qcML file as attachment to the quality parameter of a certain run or set. Thus, it needs as parameters the CV accession of the quality parameter to which it should to be attached to (*qp_att_acc*), and a CV accession for the content itself (*cv_acc*) if there is one available. The image from the ImageCreator we will attach to *QC:0000004*, which is the *MS aquisition result details* accession. If a QualityParameter with that accession does not exist, it will be created for that purpose. And we give itself the CV *QC:0000055* which is the *MS experiment heatmap* accession. We will not specify a run, since in a single run QC we are sure there is just one run to consider and the QCEmbedder will figure that out.

4.3.1 ID ratio

Next, we will have a look at the ID ratio. Therefore, import the qc_id_ratio workflow from the workflow folder and copy that meta node into your workflow to take input from the

QCEmbedder. The ID Ratio meta node will create a plot of the measured spectra vs. the identified spectra in a M/Z vs. RT map. If you open the meta node, you will see that we can extract plots or tables just similar to how we could embed them. But this time we handle a table, which will be extracted in csv format. With dedicated KNIME reader nodes, we read the tables into KNIME and calculate the metric with R. It will map the recorded MS2 spectra against the Identifications via the RT and M/Z coordinates of their precursors.
The resulting plot will be included in the qcML file and we can move to the next metric.

4.3.2 Total Ion Current

Now, we will embed a plot of the total ion current (TIC). As before, import the qc_tic_plot workflow and copy the metanode into your workflow to recieve the input qcML from the ID ration meta node. The operation is pretty straight-forward and will yield the total ion current measured on MS1 level.

4.3.3 Mass Accuracy

To take a closer look at our identifications mass accuracy, we import the qc_mass_accuracy workflow and copy the meta node to recieve input from the last meta node. This time, two plots are being generated. A histogram will show you the distribution of your identification errors. It should resemble a narrow gaussian distribution.
The other plot will show you the identification errors over time (RT).
Both plots will have the smoothed function of the error plotted in red.

4.3.4 Fractional Mass

Our last plot will be a look on what features we found. Import and copy the metanode from the qc_fractional_mass workflow. Also, copy the extra input node. This input will be an external reference file containing all the potential theoretical masses we could see with the given experiment (🗀 `QC` ▸ `theoretical_masses.txt`). It will plot these on nominal vs. fractional mass in blue and the found feature centroids in red. So a real peptide feature should be located inside these blue clouds.

4.3.5 Final preparations

After we have all the plots embedded and calculated all the QC values we want, we can get rid of the verbose or residual data we were collecting in order to be able to calculate everything we want. The QCShrinker node from Community Nodes > OpenMS > Utilities will take care of that.

4.4 Inspect the quality reports

As we have a ZipLoop in our labelfree quantification workflow, creating a QC report of several runs is easy. If you connect the last QC node with a new input port of the ZipLoopEnd node, you can take the respective ouput port and connect it with a new Output Folder, select a destination and have a look at the QC reports by opening the qcML files in a browser.

The final workflow, as it should be build in this section, can be seen in Figure 9.

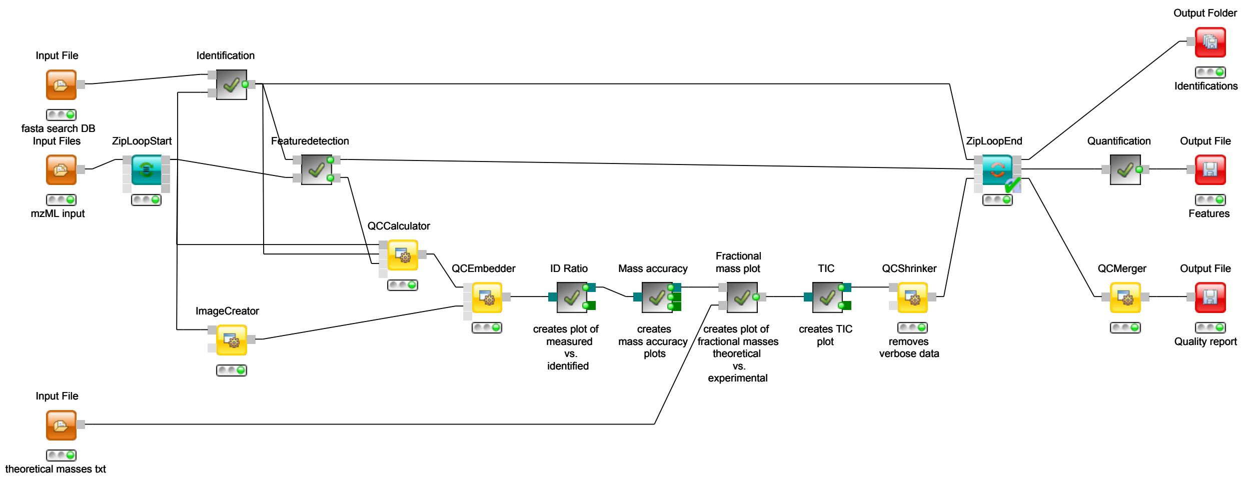


Figure 9: Complete quality control workflow.

5 Metabolomics

5.1 Introduction

Quantitation and identification of chemical compounds are basic tasks in metabolomic studies. In this tutorial session we construct a UPLC-MS based, label-free quantitation and identification workflow. Following quantitation and identification we then perform statistical downstream analysis to detect quantitation values that differ significantly between two conditions. This approach can, for example, be used to detect biomarkers. Here, we use two spike-in conditions of a dilution series (0.5 mg/l and 10.0 mg/l, male blood background, measured in triplicates) comprising seven isotopically labeled compounds. Goal of this tutorial is to detect and quantify these differential spike-in compounds against the complex background.

5.2 Quantifying metabolites across several experiments

For the quantification of the metabolites we choose a similar approach to the one used for peptides based on one of OpenMS' feature finder.

- Create a new workflow called for instance "Metabolomics".
- Add a `Input Files` node and configure it with all mzML files from `Metabolomics` ▸ `datasets`.
- Add a `ZipLoopStart` node and connect the `Input Files` node to the first port of the `ZipLoopStart` node.
- Add a `FeatureFinderMetabo` node (from Community Nodes › OpenMS › Quantitation and connect the first output port of the `ZipLoopStart` to the `FeatureFinderMetabo`.
- For an optimal result adjust the following settings. Please note that some of these are advanced parameters.

parameter	**value**
algorithm → common → chrom_fwhm	8.0
algorithm → mtd → trace_termination_criterion	sample_rate
algorithm → mtd → min_trace_length	3.0
algorithm → mtd → max_trace_length	600.0
algorithm → epd → width_filtering	off

- Add a `ZipLoopEnd` node and connect the output of the `FeatureFinderMetabo` to the first port of the `ZipLoopEnd` node.

- After the `ZipLoopEnd` node add a `MapAlignerPoseClustering` node (Community Nodes > OpenMS > Map Alignment), set its Output Type to featureXML, and adjust the following settings

parameter	**value**
algorithm → max_num_peaks_considered	-1
algorithm → superimposer → mz_pair_max_distance	0.005
algorithm → superimposer → num_used_points	10000
algorithm → pairfinder → distance_RT → max_difference	20.0
algorithm → pairfinder → distance_MZ → max_difference	20.0
algorithm → pairfinder → distance_MZ → unit	ppm

- After the `MapAlignerPoseClustering` add a `FeatureLinkerUnlabeledQT` (Community Nodes > OpenMS > Map Alignment) and adjust the following settings

parameter	**value**
algorithm → distance_RT → max_difference	40.0
algorithm → distance_MZ → max_difference	20.0
algorithm → distance_MZ → unit	ppm

- After the `FeatureLinkerUnlabeledQT` add a `TextExporter` node (Community Nodes > OpenMS > File Handling).

- Add an `Output Folder` node and configure it with an output directory where you want to store the resulting files.

- Run the pipeline and inspect the output.

You should find a single, tab-separated file containing the information on where metabolites were found and with which intensities. You can also add `Output Folder` nodes at different stages of the workflow and inspect the intermediate results (e.g., identified metabolite features for each input map). The complete workflow can be seen in Figure 10. In the following section we will try to identify those metabolites.

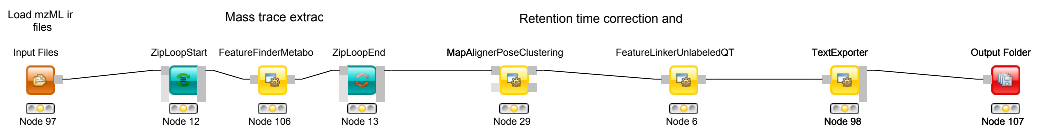


Figure 10: Label-free quantification workflow for metabolites

5.3 Identifying metabolites in LC-MS/MS samples

At the current state we found several metabolites in the individual maps but so far don't know what they are. To identify metabolites OpenMS provides the `AccurateMassSearch` node which searches observed masses against the Human Metabolome Database (HMDB)[7, 8, 9]. We start with the workflow from the previous section (see Figure 10).

- Add a `FileConverter` node and connect the output of the `FeatureLinkerUnlabeledQT` to the incoming port.
- Open the Configure dialog of the `FileConverter` and select the tab "OutputTypes". In the drop down list for FileConverter.1.out select "featureXML".
- Add an `AccurateMassSearch` node and connect the output of the `FileConverter` to the first port of the `AccurateMassSearch`.
- Add four `Input File` nodes and configure them with the following files
  - `Metabolomics▸databases▸PositiveAdducts.tsv`
    This file specifies the list of adducts that are considered in the positive mode. Each line contains the formula and charge of an adduct separated by a semicolon (e.g. M+H;1+). The mass of the adduct is calculated automatically.

  - 🗀 `Metabolomics ▸ databases ▸ NegativeAdducts.tsv`
    This file specifies the list of adducts that are considered in the negative mode analogous to the positive mode.
  - 🗀 `Metabolomics ▸ databases ▸ HMDBMappingFile.tsv`
    This file contains information from a metabolite database in this case from HMDB. It has three tab-separated columns mass, formula, and identifier. This allows for an efficient search by mass.
  - 🗀 `Metabolomics ▸ databases ▸ HMDB2StructMapping.tsv`
    This file contains additional information about the identifiers in the mapping file. It has four tab-separated columns that contain the identifier, name, SMILES, and INCHI. These will be included in the result file. The identifiers in this file must match the identifiers in the HMDBMappingFile.tsv.

- In the same order as they are given above connect them to the remaining input ports of the `AccurateMassSearch` node.
- Add an `Output Folder` node and connect the first output port of the `AccurateMassSearch` node to the output folder.

The result of the `AccurateMassSearch` node is in the mzTab format [10] so you can easily open it in a text editor or import it into Excel or KNIME, which we will do in the next section. The complete workflow from this section is shown in Figure 11.

5.4 Convert your data into a KNIME table

The result from the `TextExporter` node as well as the result from the `AccurateMassSearch` node are files while standard KNIME nodes display and processes only KNIME tables. To convert these files into KNIME tables we need two different nodes. For the `AccurateMassSearch` results we use the `SmallMoleculeMzTabReader` node (Community Nodes › OpenMS › Conversion › mzTab), for the result of the `TextExporter` we use the `ConsensusTextReader` (Community Nodes › OpenMS › Conversion).

When executed, both nodes will import the OpenMS files and provide access to the data as KNIME tables. You can now easily combine both tables using the `Joiner` node

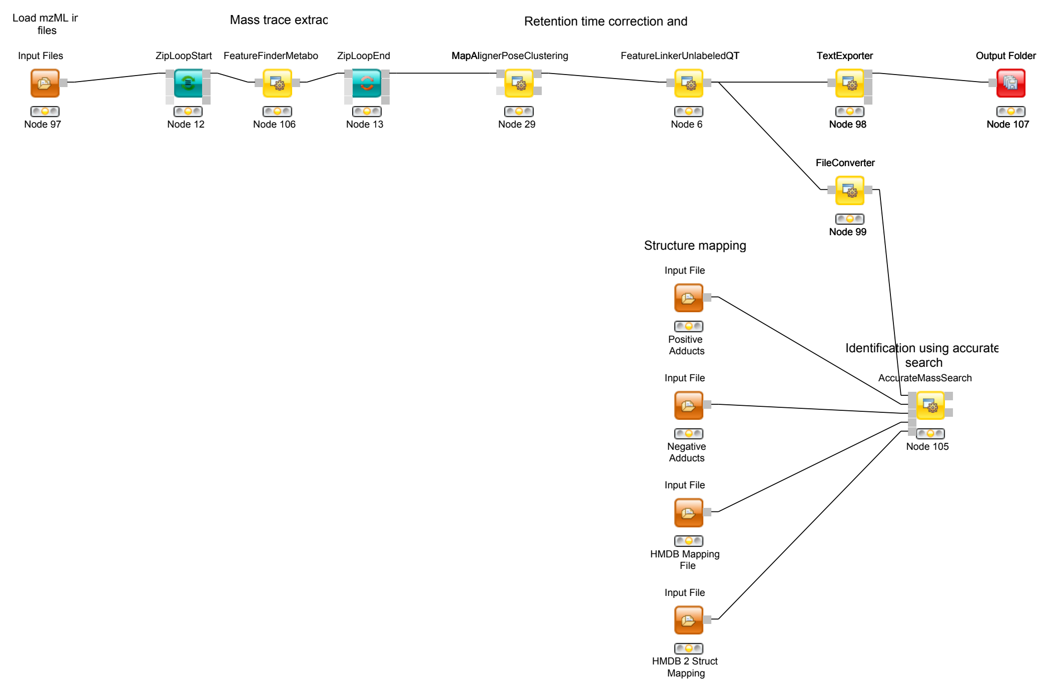


Figure 11: Label-free quantification and identification workflow for metabolites

(Data Manipulation > Column > Split & Combine) and configuring it to match the m/z and retention time values of the respective tables. The full workflow is shown in Figure 12.

5.4.1 Bonus task: Visualising data

Now that you have your data in KNIME you should try to get a feeling for the capabilities of KNIME.

Task

Check out the `Molecule Type Cast` node to render the structural formula contained in the result table.

Task

Have a look at the `Column Filter` node to reduce the table to the interesting columns, e.g., only the Ids, chemical formula, and intensities.

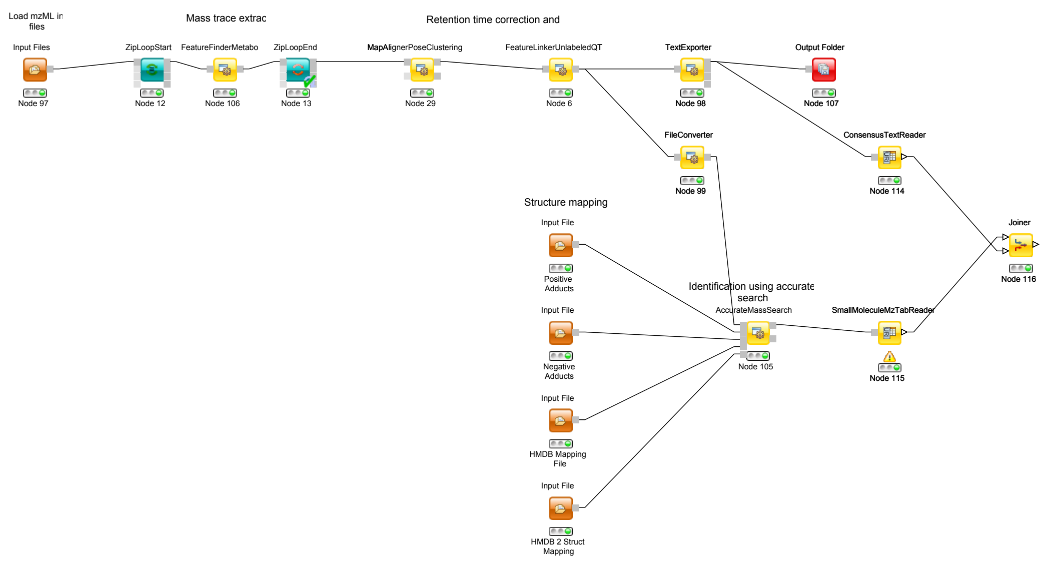


Figure 12: Label-free quantification and identification workflow for metabolites that loads the results into KNIME and joins the tables.

Task

✓ Try to compute and visualise the m/z and retention time error of the different elements of the consensus features.

5.5 Downstream data analysis and reporting

In this part of the metabolomics session we take a look at more advanced downstream analysis and the use of the statistical programming language R. As laid out in the introduction we try to detect a set of spike-in compounds against a complex blood background. As there are many ways to perform this type of analysis we provide a complete workflow.

Task

✓ Import the workflow from `Workflows` ▸ `metabolite_analysis.zip` in KNIME: File ❯ Import KNIME Workflow...

The section below will guide you in your understanding of the different parts of the workflow. Once you understood the workflow you should play around and be creative.

Maybe create a novel visualization in KNIME or R? Do some more elaborate statistical analysis? Feel free to experiment and show us your results if you like. Note that some basic R knowledge is required to fully understand the processing in `R Snippet` nodes.

5.5.1 Data preparation ID

This part is analogous to what you did for the simple metabolomics pipeline.

5.5.2 Data preparation Quant

The first part is identical to what you did for the simple metabolomics pipeline. Additionally, we convert zero intensities into NA values and remove all rows that contain at least one NA value from the analysis. We do this using a very simple `R Snippet` and subsequent `Missing Value filter` node.

Task

✓ Inspect the `R Snippet` by double-clicking on it. The KNIME table that is passed to an `R Snippet` node is available in R as a data.frame named knime.in. The result of this node will be read from the data.frame knime.out after the script finishes. Try to understand and evaluate parts of the script (Eval Selection). In this dialog you can also print intermediary results using for example the R command head() or cat() to the Console pane.

5.5.3 Statistical analysis

After we linked features across all maps, we want to identify features that are significantly deregulated between the two conditions. We will first scale and normalize the data, then perform a t-test, and finally correct the obtained p-values for multiple testing using Benjamini-Hochberg. All of these steps will be carried out in individual `R Snippet` nodes.

- Double-click on the first `R Snippet` node labeled "log scaling" to open the `R Snippet` dialog. In the middle you will see a short R script that performs the log scaling. To perform the log scaling we use a so-called regular expression (grepl) to select all columns containing the intensities in the six maps and take the log_2 logarithm.

- The output of the log scaling node is also used to draw a boxplot that can be used to examine the structure of the data. Since we only want to plot the intensities in the different maps (and not m/z or rt) we first use a `Column Filter` node to keep only the columns that contain the intensities. We connect the resulting table to a `Box Plot` node which draws one box for every column in the input table. Right-click and select View: Box Plot.

- The median normalization is performed in a similar way to the log scaling. First we calculate the median intensity for each intensity column, then we subtract the median from every intensity.

- Open the `Box Plot` connected to the normalization node and compare it to the box plot connected to the log scaling node to examine the effect of the median normalization.

- To perform the t-test we defined the two groups we want to compare. Then we call the t-test for every consensus feature unless it has missing values. Finally we save the p-values and fold-changes in two new columns named p-value and FC.

- The `Numeric Row Splitter` is used to filter less interesting parts of the data. In this case we only keep columns where the fold-change is ≥ 2.

- We adjust the p-values for multiple testing using Benjamini-Hochberg and keep all consensus features with a q-value ≤ 0.01 (i.e. we target a false-discovery rate of 1%).

5.5.4 Data preparation for Reporting

Following the identification, quantification and statistical analysis our data is merged and formatted for reporting. First we want to discard our normalized and logarithmized intensity values in favor of the original ones. To this end we first remove the intensity columns (`Column Filter`) and add the original intensities back (`Joiner`). Note that we use an *Inner Join* [1]. Combining ID and Quantification table into a single table is again achieved using a `Joiner` node.

[1] *Inner Join* is a technical term that describes how database tables are merged.

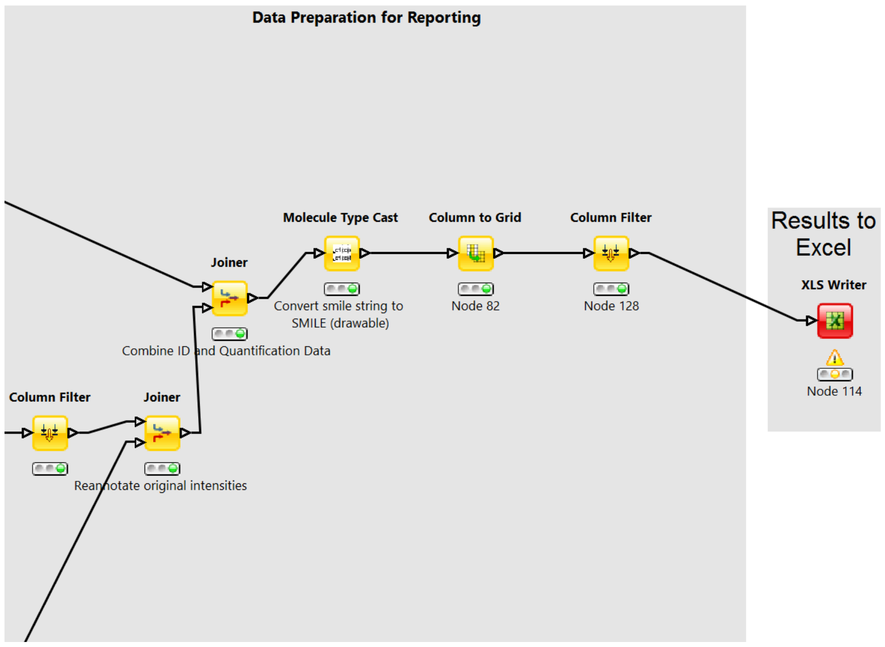

Data Preparation for Reporting
Molecule Type Cast
Column to Grid
Column Filter
Results to Excel
Joiner
XLS Writer
Convert smile string to SMILE (drawable)
Node 82
Node 128
Node 114
Combine ID and Quantification Data
Column Filter
Joiner
Reannotate original intensities

Figure 13: Data preparation for reporting

Question

What happens if we use an *Left Outer Join*, *Right Outer Join* or *Full Outer Join* instead of the *Inner Join*?

Task

Inspect the output of the join operation and after the Molecule Type Cast.

While all relevant information is now contained in our table the presentation could be improved. Currently, we have several rows corresponding to a single consensus feature (=linked feature) but with different, alternative identifications. It would be more convenient to have only one row for each consensus feature with all accurate mass identifications added as additional columns. To achieve we use the `Column to Grid` node that flattens several rows with the same consensus number into a single one. Note that we have to specify the maximum number of columns in the grid so we set this to a large value (e.g. 100). We finally select only the columns we are interested in with the last `Column Filter` and export the data to an Excel file (`XLS Writer`).

6 OpenSWATH

6.1 Introduction

OpenSWATH [11] is a module of OpenMS that allows analysis of LC-MS/MS DIA (data independent acquisition) data using the approach described by Gillet *et al.* [12]. The DIA approach described there uses 32 cycles to iterate through precursor ion windows from 400-426 Da to 1175-1201 Da and at each step acquires a complete, multiplexed fragment ion spectrum of all precursors present in that window. After 32 fragmentations (or 3.2 seconds), the cycle is restarted and the first window (400-426 Da) is fragmented again, thus delivering complete "snapshots" of all fragments of a specific window every 3.2 seconds.

The analysis approach described by Gillet et al. extracts ion traces of specific fragment ions from all MS2 spectra that have the same precursor isolation window, thus generating data that is very similar to SRM traces.

6.2 Installation of OpenSWATH

OpenSWATH has been fully integrated since OpenMS 1.10 (`http://open-ms.sourceforge.net` [3, 2, 13]).

6.3 Installation of mProphet

mProphet (`http://www.mprophet.org/`) [14] is available as standalone script in 🗀 `External_Tools` ▸ `mProphet`. R (`http://www.r-project.org/`) and the package MASS (`http://cran.r-project.org/web/packages/MASS/`) are further required to execute mProphet. Please obtain a version for either Windows, Mac or Linux directly from CRAN.

pyprophet, a much faster reimplementation of the mProphet algorithm is available from PyPI (`https://pypi.python.org/pypi/pyprophet/`). The usage of pyprophet instead of mProphet is suggested for large-scale applications, but the installation requires more dependencies and therefore, for this tutorial the application of mProphet is described.

6.4 Generating the Assay Library

6.4.1 Generating TraML from transition lists

OpenSWATH requires the assay libraries to be supplied in the TraML format [15]. To enable manual editing of transition lists, the TOPP tool `ConvertTSVToTraML` is available that uses tab separated files as input. Example datasets are provided in 🗀`OpenSWATH`▸`assay`. Please note that the transition lists need to be named `.csv`.

The header of the transition list contains the following variables (with example values in brackets):

`PrecursorMz`
The mass-to-charge (m/z) of the precursor ion. (728.88)

`ProductMz`
The mass-to-charge (m/z) of the product or fragment ion. (924.539)

`Tr_recalibrated`
The **normalized** retention time (or iRT) [16] of the peptide. (26.5)

`transition_name`
A **unique** identifier for the transition.
(AQUA4SWATH_HMLangeA_ADSTGTLVITDPTR(UniMod:267)/2_y8)

`CE`
The collision energy that should be used for the acquisition. (27)
`Optional` (not used by OpenSWATH)

`LibraryIntensity`
The relative intensity of the transition. (3305.3)

`transition_group_id`
A **unique** identifier for the transition group.
(AQUA4SWATH_HMLangeA_ADSTGTLVITDPTR(UniMod:267)/2)

`decoy`
A binary value whether the transition is target or decoy (target:0, decoy:1). (0)

`PeptideSequence`
: The unmodified peptide sequence. (ADSTGTLVITDPTR)

`ProteinName`
: A unique identifier for the protein. (AQUA4SWATH_HMLangeA)

`Annotation`
: The fragment ion annotation. (y8)
: `Optional` (not used by OpenSWATH)

`FullUniModPeptideName`
: The peptide sequence with UniMod modifications. (ADSTGTLVITDPTR(UniMod:267))

`MissedCleavages`
: The number of missed cleavages during enzymatic digestion. (0)
: `Optional` (not used by OpenSWATH)

`Replicates`
: The number of replicates. (0)
: `Optional` (not used by OpenSWATH)

`NrModifications`
: The number of modifications. (0)
: `Optional` (not used by OpenSWATH)

`PrecursorCharge`
: The precursor ion charge. (2)

`GroupLabel`
: The stable isotope label. (light)
: `Optional` (not used by OpenSWATH)

`UniprotID`
: The Uniprot ID of the protein. ()
: `Optional` (not used by OpenSWATH)

To convert transitions lists to TraML, use `ConvertTSVToTraML`:

Linux or Mac

On the Terminal:

```
ConvertTSVToTraML –in OpenSWATH_SGS_AssayLibrary.csv –out OpenSWATH_SGS_AssayLibrary.↩
    TraML
```

Windows

On the TOPP command line:

```
ConvertTSVToTraML.exe –in OpenSWATH_SGS_AssayLibrary.csv –out OpenSWATH_SGS_AssayLibrary↩
    .TraML
```

6.4.2 Appending decoys to a TraML

To append decoys to a TraML, the TOPP tool `OpenSwathDecoyGenerator` can be used:

Linux or Mac

On the Terminal:

```
OpenSwathDecoyGenerator –in OpenSWATH_SGS_AssayLibrary.TraML –out ↩
    OpenSWATH_SGS_AssayLibrary_with_Decoys.TraML –min_transitions 3 –max_transitions 6 –↩
    method shuffle –append –exclude_similar
```

Windows

On the TOPP command line:

```
OpenSwathDecoyGenerator.exe –in OpenSWATH_SGS_AssayLibrary.TraML –out ↩
    OpenSWATH_SGS_AssayLibrary_with_Decoys.TraML –min_transitions 3 –max_transitions 6 –↩
    method shuffle –append –exclude_similar
```

6.5 OpenSWATH KNIME

An example KNIME workflow for OpenSWATH is supplied in 🗀`Workflows` (Figure 14). The example dataset can be used for this workflow (filenames in brackets):

1. Open 🗀 `Workflows` ▸ `OpenSWATH.zip` in KNIME: File ▸ Import KNIME Workflow... .
2. Select the normalized retention time (iRT) assay library in TraML format by double-clicking on node Input File ▸ iRT Assay Library.
   (🗀 `OpenSWATH` ▸ `assay` ▸ `OpenSWATH_iRT_AssayLibrary.TraML`)
3. Select the SWATH MS data in mzML format as input by double-clicking on node Input File ▸ SWATH-MS files.
   (🗀 `OpenSWATH` ▸ `data` ▸ `split_napedro_L120420_010_SW-*.nf.pp.mzML`)
4. Select the target peptide assay library in TraML format as input by double-clicking on node Input Files ▸ Assay Library.
   (🗀 `OpenSWATH` ▸ `assay` ▸ `OpenSWATH_SGS_AssayLibrary.TraML`)
5. Set the output destination by double-clicking on node Output File.
6. Run the workflow.

The resulting output can be found at your selected path, which will be used as input for mProphet. Execute the script on the Terminal (Linux or Mac) or cmd.exe (Windows) in 🗀 `OpenSWATH` ▸ `result`:

```
R --slave --args bin_dir=../../../External_Tools/mProphet/ mquest=OpenSWATH_output.csv workflow=↩
    LABEL_FREE num_xval=5 run_log=FALSE write_classifier=1 write_all_pg=1 < ../../../↩
    External_Tools/mProphet/mProphet.R
```

The main output will be called
🗀 `OpenSWATH` ▸ `result` ▸ `mProphet_all_peakgroups.xls`
with statistical information available in
🗀 `OpenSWATH` ▸ `result` ▸ `mProphet.pdf`.

Please note that due to the semi-supervised machine learning approach of mProphet the results differ slightly when mProphet is executed several times.

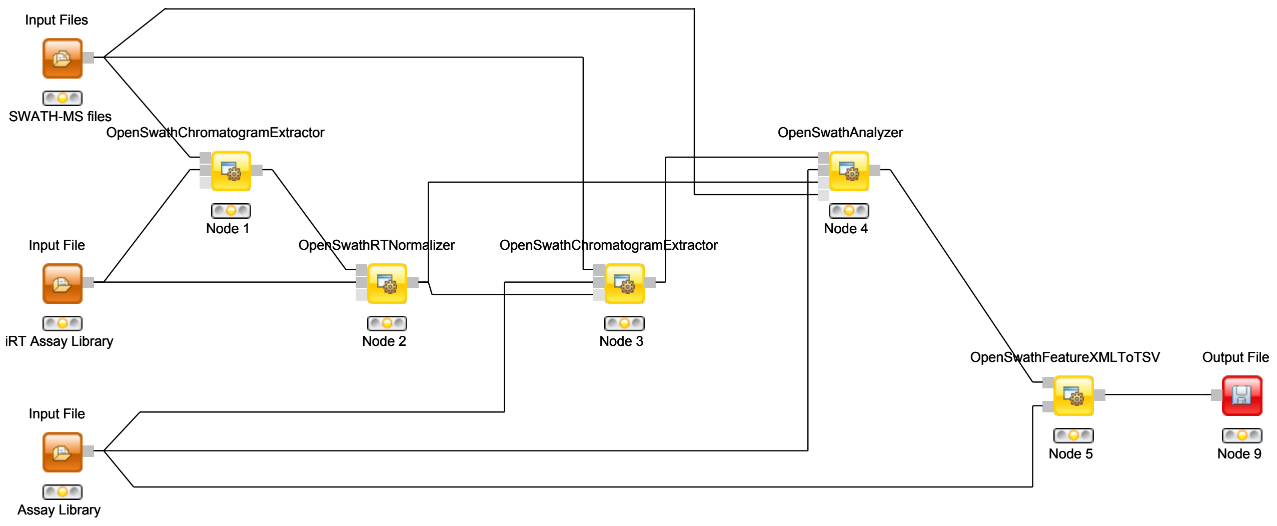


Figure 14: OpenSWATH KNIME Workflow.

6.6 Example dataset

This sample dataset is part of the larger SWATH MS Gold Standard (SGS) dataset which is described in the publication of Roest et al. [11]. It contains one of 90 SWATH MS runs with significant data reduction (peak picking of the raw, profile data) to make file transfer and working with it easier.

6.7 Real-life applications

SWATH-MS datasets are huge, several gigabyte per run. Especially when complex samples in combination with large assay libraries are analyzed, the TOPP tool based workflow requires much computational resources. For this reason, an integrated tool (`OpenSwathWorkflow`) combining all the steps in a single executable has been developed. It is shipped with OpenMS/develop and will be shipped with OpenMS 1.12. Instructions on how to use this tool can be found on `http://www.openswath.org`.

7 An introduction to pyOpenMS

7.1 Introduction

pyOpenMS provides Python bindings for a large part of the OpenMS library for mass spectrometry based proteomics. It thus provides access to a feature-rich, open-source algorithm library for mass-spectrometry based proteomics analysis. These Python bindings allow raw access to the data-structures and algorithms implemented in OpenMS, specifically those for file access (mzXML, mzML, TraML, mzIdentML among others), basic signal processing (smoothing, filtering, de-isotoping and peak-picking) and complex data analysis (including label-free, SILAC, iTRAQ and SWATH analysis tools).

pyOpenMS is integrated into OpenMS starting from version 1.11. This tutorial is addressed to people already familiar with Python. If you are new to Python, we suggest to start with a Python tutorial (`http://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_2.6`).

7.2 Installation

7.2.1 Windows

1. Install Python 2.7 (`http://www.python.org/download/`)
2. Install NumPy (`http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy`)
3. Install setuptools (`https://pypi.python.org/pypi/setuptools`)
4. On the command line:

```
easy_install pyopenms
```

7.2.2 Mac OS X 10.8

1. On the Terminal:

```
sudo easy_install pyopenms
```

7.2.3 Linux

1. Install Python 2.6 or 2.7 (Debian: python-dev, RedHat: python-devel)
2. Install NumPy (Debian / RedHat: python-numpy)
3. Install setuptools (Debian / RedHat: python-setuptools)
4. On the Terminal:

```
easy_install pyopenms
```

7.3 Build instructions

Instructions on how to build pyOpenMS can be found online (`http://ftp.mi.fu-berlin.de/OpenMS/documentation/html/pyOpenMS.html`).

7.4 Your first pyOpenMS tool: pyOpenSwathFeatureXMLToTSV

The first tool that you are going to re-implement is a TOPP tool called OpenSwathFeatureXMLToTSV. Take a look at the help of the tool:

```
OpenSwathFeatureXMLToTSV —— Converts a featureXML to a mProphet tsv.
Version: 1.11.0 Aug 16 2013, 23:41:21, Revision: 11695

Usage:
  OpenSwathFeatureXMLToTSV <options>

Options (mandatory options marked with '*'):
  —in <files>*                      Input files separated by blank (valid format
                                    s: 'featureXML')
  —tr <file>*                       TraML transition file (valid formats: 'traML
                                    ')
  —out <file>*                      Tsv output file (mProphet compatible) (valid
                                    formats: 'csv')
  –short_format                    Whether to write short (one peptide per line
                                   ) or long format (one transition per line).
  –best_scoring_peptide <varname>  If only the best scoring feature per peptide
                                   should be printed, give the variable name

Common TOPP options:
  –ini <file>                      Use the given TOPP INI file
  –threads <n>                     Sets the number of threads allowed to be
                                   used by the TOPP tool (default: '1')
  –write_ini <file>                Writes the default configuration file
  ––help                           Shows options
  ––helphelp                       Shows all options (including advanced)
```

OpenSwathFeatureXMLToTSV converts a featureXML file to a tab-separated value text file. This example will teach you how to use pyOpenMS in combination with Python to implement such a tool very quickly.

7.4.1 Basics

The first task that your tool needs to be able to do is to read the parameters from the command line and provide a main routine. This is all standard Python and no pyOpenMS is needed so far:

```
#!/usr/bin/env python
import sys

def main(options):

    # test parameter handling
    print options.infile, options.traml_in, options.outfile

def handle_args():
    import argparse

    usage = ""
    usage += "\nOpenSwathFeatureXMLToTSV –– Converts a featureXML to a mProphet tsv."

    parser = argparse.ArgumentParser(description = usage )
    parser.add_argument('––in', dest="infile", help = 'An input file containing features [↩
        featureXML]')
    parser.add_argument("––tr", dest="traml_in", help="An input file containing the transitions [↩
        TraML]")
    parser.add_argument("––out", dest="outfile", help="Output mProphet TSV file [tsv]")

    args = parser.parse_args(sys.argv[1:])
    return args

if __name__ == '__main__':
    options = handle_args()
    main(options)
```

Execute this code in the example script

`./pyOpenMS/OpenSwathFeatureXMLToTSV_basics.py`

```
python OpenSwathFeatureXMLToTSV_basics.py --help
usage: OpenSwathFeatureXMLToTSV_basics.py [-h] [--in INFILE] [--tr TRAML_IN]
                                          [--out OUTFILE]

OpenSwathFeatureXMLToTSV -- Converts a featureXML to a mProphet tsv.

optional arguments:
  -h, --help     show this help message and exit
  --in INFILE    An input file containing features [featureXML]
  --tr TRAML_IN  An input file containing the transitions [TraML]
  --out OUTFILE  Output mProphet TSV file [tsv]
```

```
python OpenSwathFeatureXMLToTSV_basics.py --in data/example.featureXML --tr assay/↩
    OpenSWATH_SGS_AssayLibrary.TraML --out example.tsv
data/example.featureXML assay/OpenSWATH_SGS_AssayLibrary.TraML example.tsv
```

The parameters are being read from the command line by the function handle_args() and given to the main() function of the script, which prints the different variables.

7.4.2 Loading data structures with pyOpenMS

Now we're going to import the pyOpenMS module with `import pyopenms` in the header of the script and load the featureXML:

```
#!/usr/bin/env python
import pyopenms
import sys

def main(options):
    # load featureXML
    features = pyopenms.FeatureMap()
    fh = pyopenms.FileHandler()
    fh.loadFeatures(options.infile, features)

    keys = []
    features[0].getKeys(keys)
    print keys

def handle_args():
    import argparse

    usage = ""
    usage += "\nOpenSwathFeatureXMLToTSV —— Converts a featureXML to a mProphet tsv."

    parser = argparse.ArgumentParser(description = usage )
    parser.add_argument('——in', dest="infile", help = 'An input file containing features [↩
        featureXML]')
    parser.add_argument("——tr", dest="traml_in", help="An input file containing the transitions [↩
        TraML]")
    parser.add_argument("——out", dest="outfile", help="Output mProphet TSV file [tsv]")

    args = parser.parse_args(sys.argv[1:])
    return args

if __name__ == '__main__':
    options = handle_args()
    main(options)
```

The function pyopenms.FeatureMap() initializes an OpenMS FeatureMap data structure. The function pyopenms.FileHandler() prepares a filehandler with the variable name fh and fh.loadFeatures(options.infile, features) takes the filename and imports the featureXML into the FeatureMap data structure.

In the next step, we're accessing the keys using the function getKeys() and printing them to stdout:

```
python OpenSwathFeatureXMLToTSV_datastructures1.py ——in data/example.featureXML ——tr assay/↩
    OpenSWATH_SGS_AssayLibrary.TraML ——out example.tsv
```

```
['PeptideRef', 'leftWidth', 'rightWidth', 'total_xic', 'peak_apices_sum', 'var_xcorr_coelution', ↩
    'var_xcorr_coelution_weighted ', 'var_xcorr_shape', 'var_xcorr_shape_weighted', '↩
    var_library_corr', 'var_library_rmsd', 'var_library_manhattan', 'var_library_dotprod', '↩
    delta_rt', 'assay_rt', 'norm_RT', 'rt_score', 'var_norm_rt_score', 'var_intensity_score', '↩
    nr_peaks', 'sn_ratio', 'var_log_sn_score', 'var_elution_model_fit_score', '↩
    xx_lda_prelim_score', 'var_isotope_correlation_score', 'var_isotope_overlap_score', '↩
    var_massdev_score', 'var_massdev_score_weighted', 'var_bseries_score', 'var_yseries_score', ↩
    'var_dotprod_score', 'var_manhatt_score', 'main_var_xx_swath_prelim_score', 'PrecursorMZ', '↩
    xx_swath_prelim_score']
```

In the next task, please load the TraML into an OpenMS TargetedExperiment data structure, analogously to the featureXML. You might want to consult the pyOpenMS manual (`http://proteomics.ethz.ch/pyOpenMS_Manual.pdf`), which provides an overview of all functionality. If you have trouble reading the TraML, search for TraMLFile(). If you can't solve the task, take a look at OpenSwathFeatureXMLToTSV_datastructures2.py

7.4.3 Converting data in the featureXML to a TSV

Now that all data structures are populated, we need to access the data using the provided API and store it in something that is directly accessible from Python. We prepared two functions for you: get_header() & convert_to_row():

```
def get_header(features):
    keys = []
    features[0].getKeys(keys)
    header = [
        "transition_group_id",
        "run_id",
        "filename",
        "RT",
        "id",
        "Sequence" ,
        "FullPeptideName",
        "Charge",
        "m/z",
        "Intensity",
        "ProteinName",
        "decoy"]
    header.extend(keys)
    return header
```

get_header() takes as input a FeatureMap and uses the getKeys() function that you have seen before to extend a predefined header list based on the contents of the FeatureMap.

The return variable is a native Python list.

```python
def convert_to_row(first, targ, run_id, keys, filename):
    peptide_ref = first.getMetaValue("PeptideRef")
    pep = targ.getPeptideByRef(peptide_ref)
    full_peptide_name = "NA"
    if (pep.metaValueExists("full_peptide_name")):
      full_peptide_name = pep.getMetaValue("full_peptide_name")

    decoy = "0"
    peptidetransitions = [t for t in targ.getTransitions() if t.getPeptideRef() == peptide_ref]
    if len(peptidetransitions) > 0:
        if peptidetransitions[0].getDecoyTransitionType() == pyopenms.DecoyTransitionType().DECOY↩
            :
            decoy = "1"
        elif peptidetransitions[0].getDecoyTransitionType() == pyopenms.DecoyTransitionType().↩
            TARGET:
            decoy = "0"

    protein_name = "NA"
    if len(pep.protein_refs) > 0:
      protein_name = pep.protein_refs[0]

    row = [
        first.getMetaValue("PeptideRef"),
        run_id,
        filename,
        first.getRT(),
        first.getUniqueId(),
        pep.sequence,
        full_peptide_name,
        pep.getChargeState(),
        first.getMetaValue("PrecursorMZ"),
        first.getIntensity(),
        protein_name,
        decoy
    ]

    for k in keys:
        row.append(first.getMetaValue(k))

    return row
```

convert_to_row() is a bit more complicated and takes as first input a Feature OpenMS class. From this, we access stored values using the provided functions (getRT(), getUniqueId(), etc). It further takes a TargetedExperiment to access information from the TraML with the provided routines. This data is then stored in a standard Python list with the variable name row and returned.

7.4.4 Putting things together

Now put these two functions into the header of

`OpenSwathFeatureXMLToTSV_datastructures2.py`.

Your final goal is to implement the conversion functionality into the main function using get_header() & convert_to_row() and to write a TSV using the standard csv module from Python `http://docs.python.org/2/library/csv.html`. Compare the results with `./result/example.tsv`. Are the results identical? Congratulations to your first pyOpenMS tool!

Hint: If you struggle at any point, take a look at OpenSwathFeatureXMLToTSV_solution.py.

7.4.5 Bonus task

Task

✓ Implement all other 142 TOPP tools using pyOpenMS.

References

[1] OpenMS, OpenMS home page [online]. 6

[2] M. Sturm, A. Bertsch, C. Gröpl, A. Hildebrandt, R. Hussong, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, A. Zerck, K. Reinert, and O. Kohlbacher, OpenMS - an open-source software framework for mass spectrometry., BMC bioinformatics **9**(1) (2008), `doi:10.1186/1471-2105-9-163`. 6, 40

[3] O. Kohlbacher, K. Reinert, C. Gröpl, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, and M. Sturm, TOPP–the OpenMS proteomics pipeline., Bioinformatics **23**(2) (Jan. 2007). 6, 40

[4] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel, KNIME: The Konstanz Information Miner, in *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*, Springer, 2007. 6

[5] M. Sturm and O. Kohlbacher, TOPPView: An Open-Source Viewer for Mass Spectrometry Data, Journal of proteome research **8**(7), 3760–3763 (July 2009), `doi:10.1021/pr900171m`. 6

[6] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. Xu, D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant, Open mass spectrometry search algorithm, Journal of Proteome Research **3**(5), 958–964 (2004). 18

[7] D. S. Wishart, D. Tzur, C. Knox, et al., HMDB: the Human Metabolome Database, Nucleic Acids Res **35**(Database issue), D521–6 (Jan 2007), `doi:10.1093/nar/gkl923`. 33

[8] D. S. Wishart, C. Knox, A. C. Guo, et al., HMDB: a knowledgebase for the human metabolome, Nucleic Acids Res **37**(Database issue), D603–10 (Jan 2009), `doi:10.1093/nar/gkn810`. 33

[9] D. S. Wishart, T. Jewison, A. C. Guo, M. Wilson, C. Knox, et al., HMDB 3.0–The Human Metabolome Database in 2013, Nucleic Acids Res **41**(Database issue), D801–7 (Jan 2013), `doi:10.1093/nar/gks1065`. 33

[10] J. Griss, A. R. Jones, T. Sachsenberg, M. Walzer, L. Gatto, J. Hartler, G. G. Thallinger, R. M. Salek, C. Steinbeck, N. Neuhauser, J. Cox, S. Neumann, J. Fan, F. Reisinger, Q.-W. Xu, N. Del Toro, Y. Perez-Riverol, F. Ghali, N. Bandeira, I. Xenarios, O. Kohlbacher, J. A. Vizcaino, and H. Hermjakob, The mzTab Data Exchange Format: communicating MS-based proteomics and metabolomics experimental results to a wider audience, Mol Cell Proteomics (Jun 2014), doi:10.1074/mcp.O113.036681. 34

[11] H. L. Röst, G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinovic, O. T. Schubert, W. Wolski, B. C. Collins, J. Malmstrom, L. Malmström, and R. Aebersold, OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data, Nature Biotechnology **32**(3), 219–223 (Mar. 2014). 40, 45

[12] L. C. Gillet, P. Navarro, S. Tate, H. Röst, N. Selevsek, L. Reiter, R. Bonner, and R. Aebersold, Targeted Data Extraction of the MS/MS Spectra Generated by Data-independent Acquisition: A New Concept for Consistent and Accurate Proteome Analysis., Molecular & Cellular Proteomics **11**(6) (June 2012), doi:10.1074/mcp.O111.016717. 40

[13] A. Bertsch, C. Gröpl, K. Reinert, and O. Kohlbacher, OpenMS and TOPP: open source software for LC-MS data analysis., Methods in molecular biology (Clifton, N.J.) **696**, 353–367 (2011), doi:10.1007/978-1-60761-987-1_23. 40

[14] L. Reiter, O. Rinner, P. Picotti, R. Huttenhain, M. Beck, M.-Y. Brusniak, M. O. Hengartner, and R. Aebersold, mProphet: automated data processing and statistical validation for large-scale SRM experiments, Nature Methods **8**(5), 430–435 (May 2011), doi:10.1038/nmeth.1584. 40

[15] E. W. Deutsch, M. Chambers, S. Neumann, F. Levander, P.-A. Binz, J. Shofstahl, D. S. Campbell, L. Mendoza, D. Ovelleiro, K. Helsens, L. Martens, R. Aebersold, R. L. Moritz, and M.-Y. Brusniak, TraML—A Standard Format for Exchange of Selected Reaction Monitoring Transition Lists, Molecular & Cellular Proteomics **11**(4) (Apr. 2012), doi:10.1074/mcp.R111.015040. 41

[16] C. Escher, L. Reiter, B. MacLean, R. Ossola, F. Herzog, J. Chilton, M. J. MacCoss, and O. Rinner, Using iRT, a normalized retention time for more targeted measurement of peptides., Proteomics **12**(8), 1111–1121 (Apr. 2012), doi:10.1002/pmic.201100463. 41