
OpenMS

Release 3.0.0

OpenMS Team

Jul 20, 2023

INTRODUCTION

1	Contents	3
1.1	What is OpenMS?	3
1.2	Background	3
1.3	Entry Points to OpenMS	15
1.4	Installation	16
1.5	OpenMS Graphical User Interfaces	29
1.6	TOPP Tools	29
1.7	UTILS Tools	32
1.8	Command Line Interface	33
1.9	TOPPView: Visualize with OpenMS	34
1.10	Recommended Workflow Systems	51
1.11	OpenMS in KNIME	53
1.12	OpenMS in Nextflow	148
1.13	OpenMS on Galaxy	149
1.14	User Quickstart Guide	150
1.15	OpenMS User Tutorial	152
1.16	Worked Examples: Different OpenMS Methods to Achieve the Same Outcome	245
1.17	OpenMS C++ Core Library	249
1.18	Build From Source	249
1.19	External Code using OpenMS	250
1.20	Developer Tutorial	252
1.21	Contributor's Quick Start Guide	252
1.22	OpenMS Git Workflow	254
1.23	Write and Label GitHub Issues	258
1.24	Adding New Tool to The TOPP suite	259
1.25	Pull Request Checklist	261
1.26	Reporting Bugs and Issues	262
1.27	Advanced	262
1.28	OpenMS Installers	264
1.29	Workflows	265
1.30	OpenMS Releases	265
1.31	Other Resources	266
1.32	Contributor FAQ	266
1.33	Developer FAQ	270
1.34	Contact Us	279
1.35	OpenMS Glossary	279
2	Indices and tables	285
	Index	287

OpenMS is an open-source software C++ library for *LC-MS* data management and analyses. It offers an infrastructure for rapid development of mass spectrometry related software. OpenMS is free software available under the three clause BSD license and runs under Windows, macOS, and Linux.

It comes with a vast variety of pre-built and ready-to-use tools for proteomics and metabolomics data analysis (*TOPP Tools*) as well as powerful 1D, 2D and 3D visualization (*TOPPView*).

OpenMS offers analyses for various quantitation protocols, including label-free quantitation, *SILAC*, *iTRAQ*, *TMT*, *SRM*, *SWATH*, etc.

It provides built-in algorithms for de-novo identification and database search, as well as adapters to other state-of-the-art tools like X!Tandem, *Mascot*, etc. It supports easy integration of OpenMS built tools into workflow engines like *KNIME*, Galaxy, WS-Pgrade, and *TOPPAS* via the *TOPP tools* concept and a unified parameter handling via a 'common tool description' (CTD) scheme.

Important: As part of the **Center for Integrative Bioinformatics** (CiBi) in the **German Network for Bioinformatics deNBI**, OpenMS is currently focusing the development efforts on the integration of OpenMS into KNIME. KNIME is a well-established data analysis framework that supports the generation of workflows for data analysis. Using a Common Tool Description (CTD) file which is writeable by every TOPP tool and a node generator program (*Generic KNIME Nodes*), all *TOPP tools* can be made available to run in KNIME.

With *pyOpenMS*, OpenMS offers Python bindings to a large part of the *OpenMS API* to enable rapid algorithm development. OpenMS supports the Proteomics Standard Initiative (PSI) formats for MS data. The main contributors of OpenMS are currently the Eberhard-Karls-Universität in Tübingen, the Freie Universität Berlin, and the University of Toronto.

CONTENTS

1.1 What is OpenMS?

OpenMS is an open-source software platform designed for the analysis and visualization of high-throughput mass spectrometry data. OpenMS has been designed to operate on all platforms, and provides a flexible framework for users to access a wide range of built-in tools or build their own tools using the existing functionality. These tools can be applied separately to data or be applied in sequence (as a workflow or pipeline) to mass spectrometry data. OpenMS is well established and has been used widely in [literature](#), particularly in the life sciences.

Fields such as proteomics and metabolomics require the rapid, large-scale identification, quantification and characterization of biomolecules which traditional analytical techniques struggle to offer. OpenMS has been created by a team of biologists and computer scientists to create a completely open-source solution that offers customisable tools for high-throughput processing of mass spectrometry data.

OpenMS provides a number of tools built from a C++ core library. These tools are collectively referred to as “The OpenMS PiPeline (TOPP) (formerly known as The OpenMS Proteomic Pipeline) tools. TOPP tools can be chained in a sequence to form workflows and can be applied to mass spectrometry data. Note: TOPP’s capabilities have been expanded to apply to a wide range of areas in the life sciences.

1.2 Background

Proteomics and metabolomics focus on complex interactions within biological systems; the former is centered on proteins while the latter is based on metabolites. To understand these interactions, we need to accurately identify the different biological components involved.

Liquid chromatography (LC) and *mass spectrometry* (MS) are the analytical techniques used to isolate and identify biological components in proteomics and metabolomics. LC-MS data can be difficult to analyze manually given its amount and complexity. Therefore, we need specialized software that can analyze high-throughput LC-MS data quickly and accurately.

1.2.1 Why use OpenMS

OpenMS is an open-source, C++ framework for analyzing large volumes of mass spectrometry data. It has been specially designed for analyzing high performance LC-MS data but over recent times, has been extended to analyze data generated by other techniques.

Note: OpenMS in recent times has been expanded to support a wide variety of mass spectrometry experiments. To design your analysis solution, [contact the OpenMS team](#) today.

To use OpenMS effectively, an understanding of chromatography and mass spectrometry is required as many of the algorithms are based on these techniques. This section provides a detailed explanation on LC and MS, and how they are combined to identify and quantify substances.

1.2.2 Liquid chromatography (LC)

Chromatography is a technique used by life scientists to separate molecules based on a specific physical or chemical property.

Video

For more information on chromatography, [view this video](#).

There are many types of chromatography, but this section focuses on LC as it is widely used in proteomics and metabolomics.

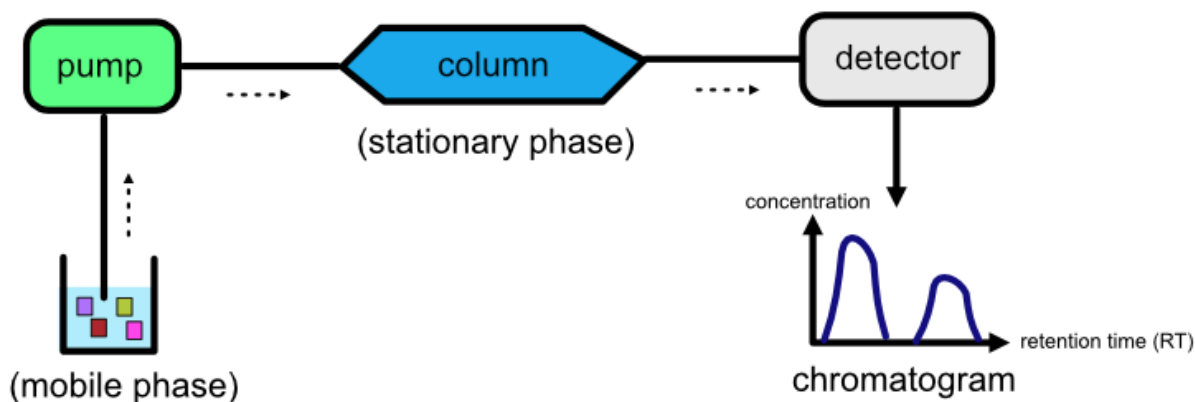
LC separates molecules based on a specific physical or chemical property by mixing a sample containing the molecules of interest (otherwise known as **analytes**) in a liquid solution.

Key components of LC

An LC setup is made up of the following components:

- A **liquid solution**, known as the **mobile phase**, containing the analytes.
- A **pump** which transports the liquid solution.
- A **stationary phase** which is a solid, homogeneous substance.
- A **column** that contains the stationary phase.
- A **detector** that plots the time it takes for the analyte to escape the column (retention time) against the analyte's concentration. This plot is called a **chromatogram**.

Refer to the image below for a diagrammatic representation of an LC setup.



How does LC work?

The liquid solution containing the analytes is pumped through a column that is attached to the stationary phase. Analytes are separated based on how strongly they interact with each phase. Some analytes will interact strongly with the mobile phase while others will be strongly attracted to the stationary phase, depending on their physical or chemical properties. The stronger an analyte's attraction is to the mobile phase, the faster it will leave the column. The time it takes for an analyte to escape from the column is called the analyte's *retention time*. As a result of their differing attractions to the mobile and stationary phases, different analytes will have different retention times, which is how separation occurs.

The retention times for each analyte are recorded by a detector. The most common detector used is the mass spectrometer, which we discuss later. However, other detection methods exist, such as:

- Light absorption (photometric detector)
- Fluorescence
- Change in diffraction index

High performance liquid chromatography (HPLC)

HPLC is the most commonly used technique for separating proteins and metabolites. In HPLC, a high-pressured pump is used to transport a liquid (solvent) containing the molecules of interest through a thin capillary column. The stationary phase is 'packed' into the column.

Video

For more information on HPLC, [view this video](#).

Several variations of HPLC exist such as:

- Reversed-phase (RP) chromatography
- Strong cation/anion exchange (SCX/SAX) chromatography
- Affinity chromatography
- Size exclusion chromatography

Special case of HPLC: Reversed-phase (RP) chromatography

RP chromatography is the most common type of HPLC with biological samples. In reversed-phase liquid chromatography, the solid phase is modified to become hydrophobic, when it is originally hydrophilic, hence the term 'reversed-phase'. The liquid phase is a mixture of water and an organic solvent. The separation of molecules happens based on the following behavior: hydrophilic analytes have a high affinity to the mobile phase and escape the column quickly while hydrophobic analytes have a high affinity towards the organic solvent and therefore, take a longer time to escape the column.

Video

For more information on RP chromatography, [view this video](#).

1.2.3 Mass spectrometry (MS)

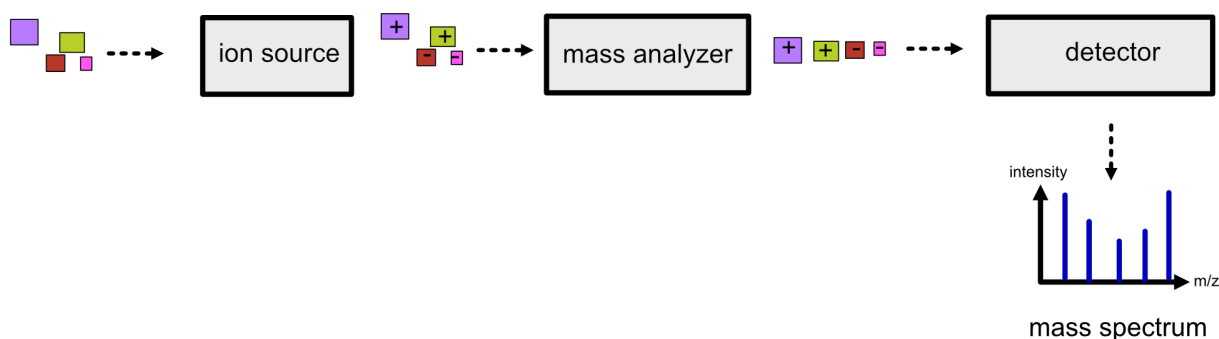
Mass spectrometry is an analytical technique used to determine the abundance of molecules in a sample.

Key components of MS

There are three key components in a mass spectrometer:

- An **ion source**, which generates *ions* from the incoming sample. All mass spectrometry techniques rely on ionized molecules to control their movement in an electric field.
- A **mass analyzer**, which separates the ions according to their mass-to-charge (m/z) ratio. There are several types such as time of flight (TOF), orbitrap and quadrupole mass analyzers. Depending on the mass analyzer, OpenMS offers calibration tools, so that highly accurate results can be achieved.
- A **detector**, which scans ions at a given time point producing a *mass spectrum*, where the intensity is plotted against the m/z .

Refer to the image below for a diagrammatic representation of the key components in MS.



Ion source

We want the analytes to move through the electrostatic and electromagnetic fields in the mass analyzer. To achieve this objective, we need to convert them to ions by charging them. There are a number of ways to charge our analytes including:

- Electrospray Ionization (ESI)
- Matrix Assisted Laser Desorption/Ionization (MALDI)
- Electron Impact Ionization (EI)

In proteomics and metabolomics, ESI and MALDI are used because they are soft ionization techniques. A soft ionization technique is one which charges analytes while keeping the molecules of interest largely intact, so that they can be characterized easily at a later stage. Hard ionization techniques such as EI shatter analytes in smaller fragments, making it difficult to characterize large molecules.

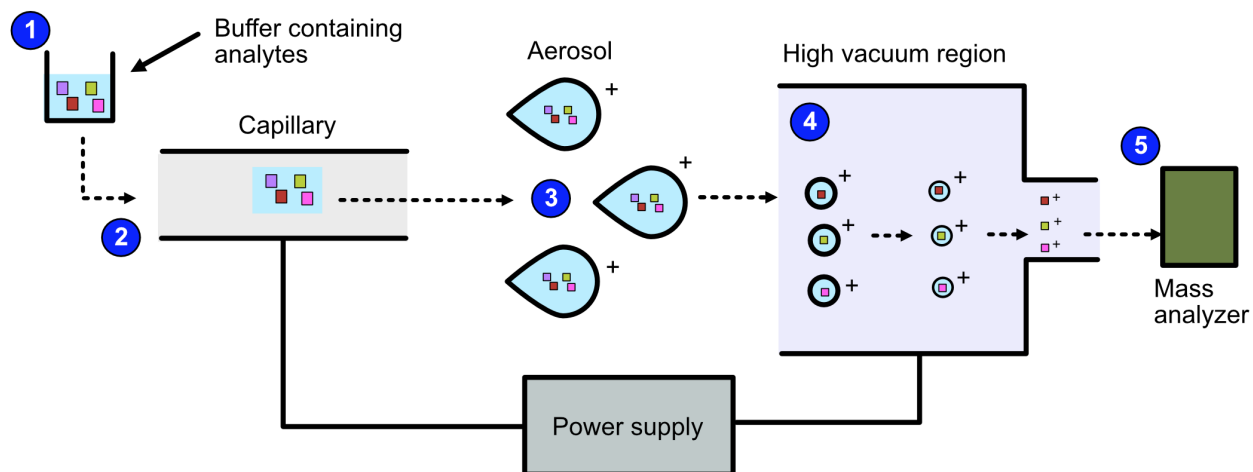
Given that OpenMS focuses on proteomic and metabolomic applications, we will describe ESI and MALDI in further detail.

Electrospray Ionization (ESI)

ESI can be broken down into the following steps.

1. The sample is dissolved in a polar, volatile buffer.
2. The sample - dissolved in the buffer - is pumped through a thin, stainless steel capillary.
3. The sample is converted to small, charged, stable droplets (aerosolized) by applying high voltage.
4. The aerosol is directed through regions of high vacuum until the droplets evaporate until only the charged molecules are left.
5. The particles are fed to the mass analyzer.

Refer to the image below for a diagrammatic representation of the steps in ESI.



Video

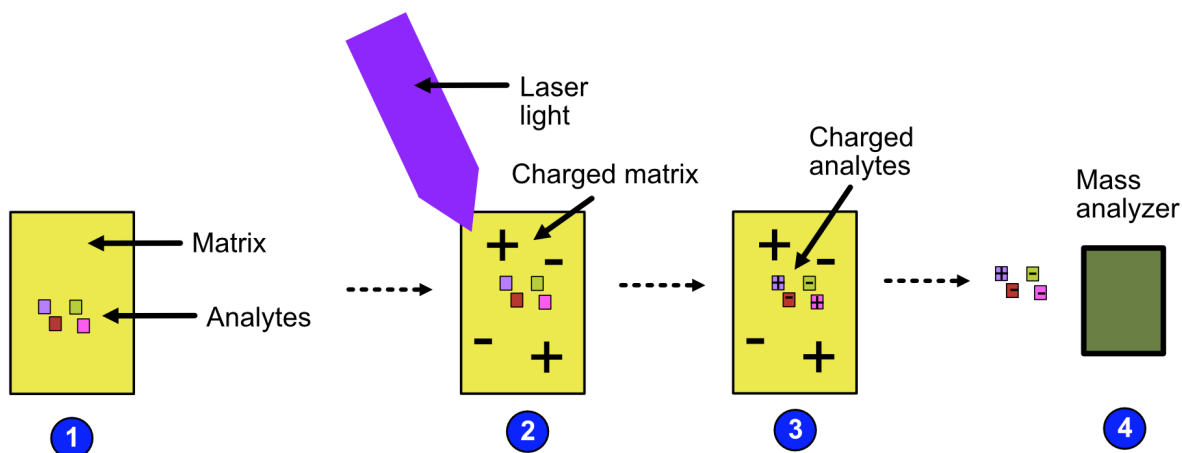
For more information on ESI, [view this video](#).

Matrix Assisted Laser Desorption/Ionization (MALDI)

MALDI can be broken down into the following steps:

1. The analytes are mixed with a small organic molecule known as a matrix.
2. The mixture is exposed to radiation with short pulses of laser light, charging the matrix.
3. The matrix transfers its charge to the analytes because the wavelength of the laser light is the same as the absorbance maximum of the matrix.
4. The analytes become charged and are fed to the mass analyzer.

Refer to the image below for a diagrammatic representation of the steps in MALDI.



Video

For more information on MALDI, [view this video](#).

Mass analyzer

Once the analytes have been charged by the ion source, we want to now sort the analytes by their mass-to-charge ratio for easy identification.

A number of mass analyzers exists. These include:

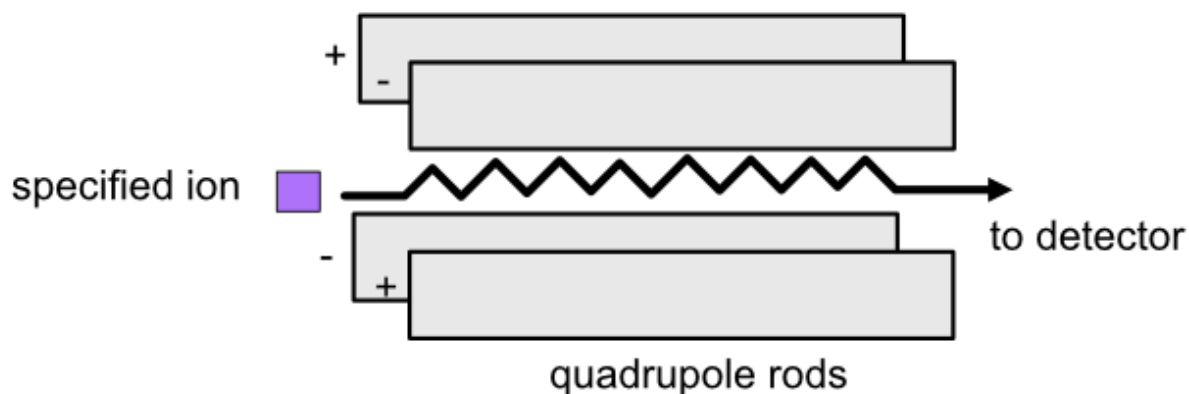
- Quadrupole analyzer
- Time-of-Flight analyzer
- Orbitrap analyzer

The next sections describe each analyzer type in detail.

Quadrupole

In a quadrupole analyzer, you can set the quadrupole voltage so that ions with a specific m/z ratio travel through. The oscillating electrostatic fields stabilize the flight path for the ions so that they can pass through the quadrupole. Other ions will be accelerated out of the quadrupole and will not make it to the end.

Refer to the image below for a diagrammatic representation of the quadrupole analyzer.



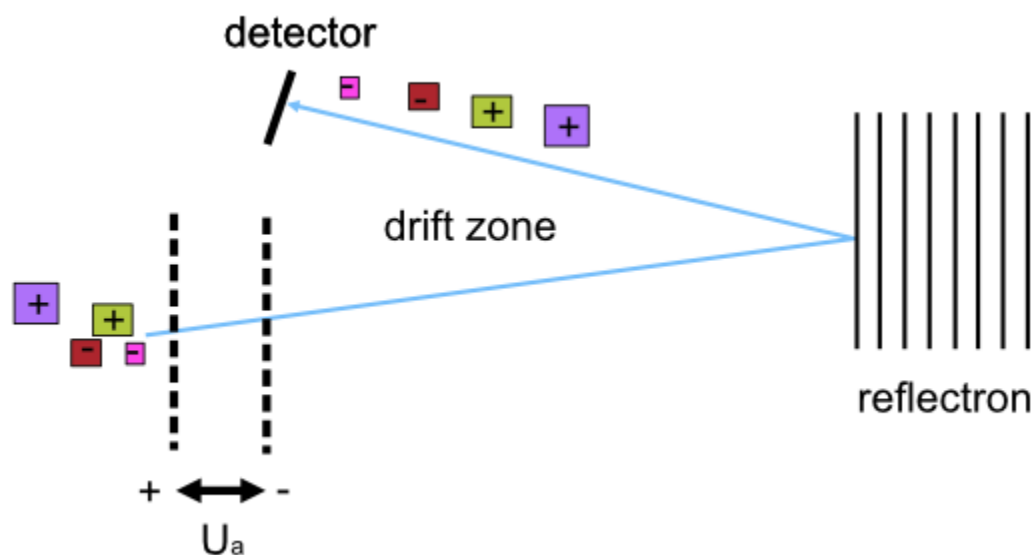
Video

For more information on quadrupole analyzers, [view this video](#).

Time-of-Flight (TOF)

In a time-of-flight analyzer, ions are extracted from the ion source through an electrostatic field in pulses in a field-free drift zone. An electrostatic mirror called a reflectron reflects the ions back onto the next component of mass spectrometry, the detector. The detector counts the particles and records the time of flight from extraction to the moment the particle hits the detector.

Refer to the image below for a diagrammatic representation of the TOF analyzer.



Lighter ions fly faster than heavier ions of the same charge and will arrive earlier at the detector. Therefore, an ion's time of flight depends on the ion's mass. The ion's time of flight is also dependant on the ion's charge. This can be demonstrated by using the following equations:

1. Potential energy is transferred to an ion with charge q accelerated by an electrostatic field with voltage.

$$E_p = qU_a(1.1)$$

2. The potential energy is converted to kinetic energy as the ion accelerates.

$$E_p = E_k = \frac{1}{2}mv^2(1.2)$$

3. We know that for a given path, s , from extraction to the detector, the time of flight, t is equal to:

$$t = \frac{s}{v}(1.3)$$

Therefore, t , for a given instrument's path length, s , depends on an ion's charge and mass.

$$t = \frac{s}{v} = \frac{s}{\sqrt{\frac{2qU_a}{m}}}(1.4)$$

Video

For more information on TOF analyzers, [view this video](#).

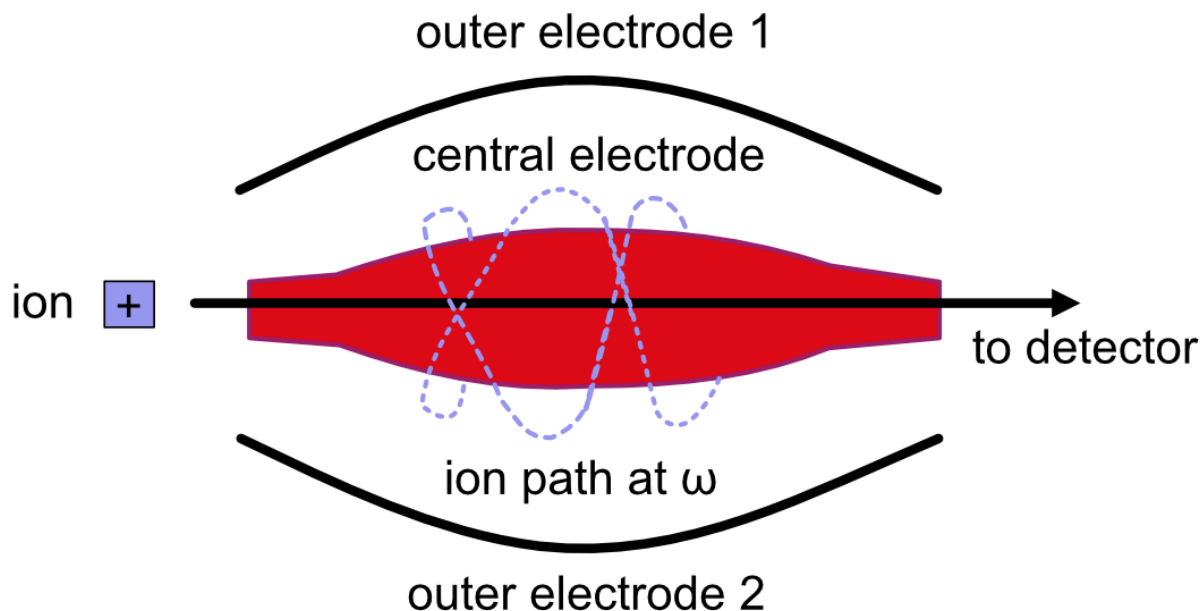
Orbitrap

The orbitrap analyzer is the most frequently used analyzer in mass spectrometry for proteomic and metabolomic applications. It consists of two outer electrodes and a central electrode. Ions are captured inside the analyzer because of an applied electrostatic field. The ions in the orbitrap analyzer oscillate around the central electrode along the axis of the electrostatic field at a set frequency, ω . This frequency is used to determine the mass-to-charge ratio using the following formula:

$$\omega = \sqrt{\frac{kz}{m}}(1.5)$$

, where k is a constant.

The following diagram is a conceptual representation of the orbitrap analyzer.



Video

For more information on orbitrap analyzers, [view this video](#).

Identifying molecules with Tandem Mass Spectrometry (MS2)

To get better results, we can use two mass analyzers sequentially to generate and analyze ions. This technique is called **tandem mass spectrometry** or MS/MS (MS2). Tandem mass spectrometry is especially useful for linear polymers like proteins, RNA and DNA.

With MS2, ions called **precursor ions** are isolated and fragmented into ion fragments or **product ions**. A *mass spectrum* is recorded for both the precursor and the product ions.

Video

For more information on MS2, [view this video](#).

Different fragmentation techniques to fragment peptides exist:

- Collision-Induced Dissociation (CID)
- Pulsed Q Dissociation (PQD)
- Electron transfer dissociation (ETD)
- Electron capture dissociation (ECD)
- Higher energy collision dissociation (HCD)

CID is the most frequently used fragmentation technique and will therefore be discussed in more detail in the following section.

Collision-induced dissociation

Collision-induced dissociation is a method to fragment peptides using an inert gas such as argon or helium. Selected primary or precursor ions enter a collision cell filled with the inert gas. The application of the inert gas on the precursor ions causes the precursor ions that reach the energy threshold to fragment into smaller, product ions and or neutral losses. A *mass spectrum* is recorded for both the precursor ions and the product ions. The *mass spectrum* for the precursor ions will give you the mass for the entire peptide while the product ions will inform you about it's amino acid composition.

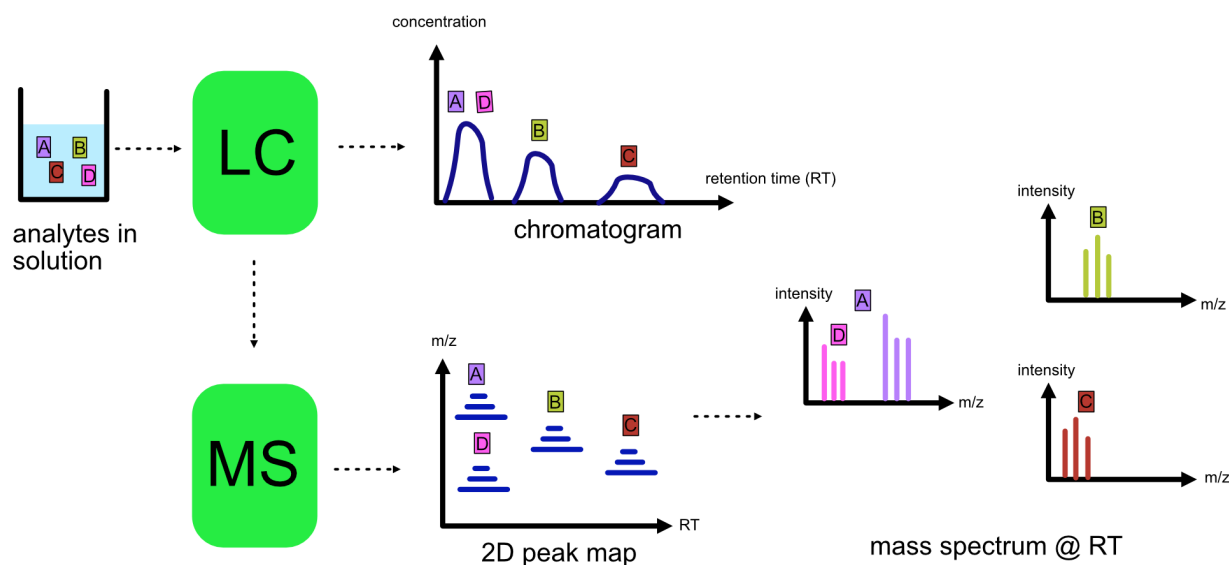
Video

For more information on CID, [view this video](#).

1.2.4 LC-MS

Liquid chromatography is often coupled with mass spectrometry to reduce complexity in the mass spectra. If complex samples were directly fed to a mass spectrometer, you would not be able to detect the less abundant analyte ions. The separated analytes from the liquid chromatography setup are directly injected into the ion source from the mass spectrometry setup. Multiple analytes that escape the column at the same time are separated by their mass-to-charge ratio using the mass spectrometer.

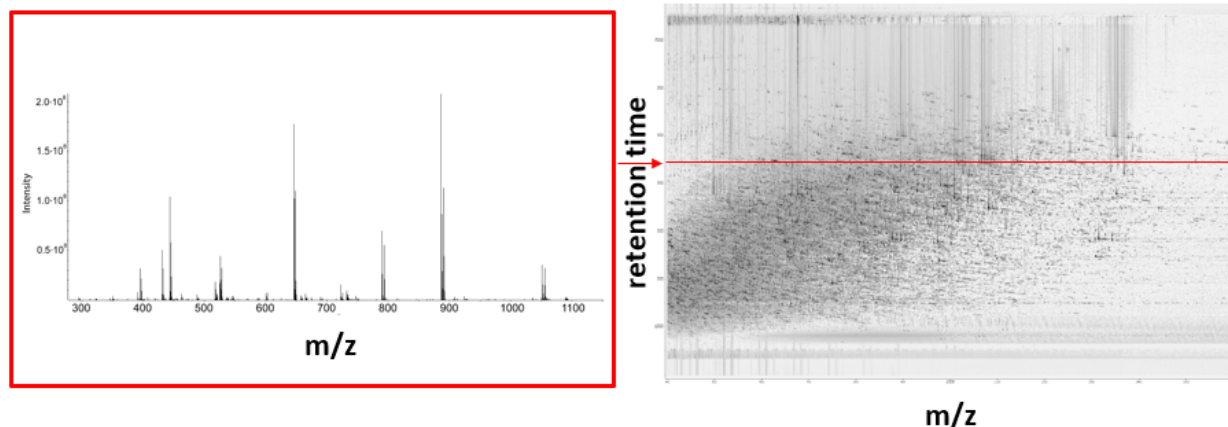
Refer to the image below for a diagrammatic representation of the LC-MS setup.



From the LC-MS setup, a set of spectra called a peak map is produced. In a peak map, each spectrum represents the ions detected at a particular retention time. Each peak in a spectrum has a retention time, mass-to-charge and intensity dimension.

From the LC-MS setup, a series of spectra are 'stacked' together to form what is known as a peak map. Each spectrum in a peak map is a collection of data points called *peaks* which indicate the retention time, mass-to-charge and intensity of each detected ion. Analyzing peak maps is difficult as different compounds can elute at the same time which means that peaks can overlap. Therefore, sophisticated techniques are required for the accurate identification and quantification of molecules.

The image below includes a spectrum at a given retention time (left) and a peak map (right).



Video

For more information on a *specific* application of LC-MS, [view this video](#).

1.2.5 Improving identification and quantification

While the combination of liquid chromatography and mass spectrometry can ease the process of characterising molecules of interest, further techniques are required to easily identify and quantify these molecules. This section discusses both labeled and label-free quantification techniques.

Labeling

Relative quantification is one strategy where one sample is chemically treated and compared to another sample without treatment. This section discusses a particular relative quantification technique called **labeling** or **stable isotope labeling** which involves the addition of isotopes to one sample. An isotope of an element behaves the same chemically but has a different mass. Stable isotope labeling is used in mass spectrometry so that scientists can easily identify proteins and metabolites.

Two types of stable isotope labeling exist: chemical labeling and metabolic labeling.

Chemical labeling

During chemical labeling, the label is attached at specific functional groups in a molecule like the N-terminus of a peptide or specific side chains.

Chemical labeling occurs late in the process, therefore experiments that incorporate this technique are not highly reproducible.

Isobaric labeling

Isobaric labeling, is a technique where peptides and proteins are labeled with chemical groups that have an identical mass, but vary in terms of distribution of heavy isotopes in their structure.

Video

For more information on isobaric labeling, view the following links:

OpenMS contains tools that analyze data from isobaric labeling experiments.

Metabolic labeling

During metabolic labeling, the organism is ‘fed’ with labeled metabolites. Metabolites include but are not limited to amino acids, nitrogen sources and glucose. Unlike chemical labeling, metabolic labeling occurs early in the study. Therefore, experiments that incorporate metabolic labeling are highly reproducible.

Stable Isotope Labeling with Amino Aids in Cell Culture (SILAC)

In SILAC, the labeled amino acids are fed to the cell culture. The labels are integrated into the proteins after a period. The labeled sample is then compared with the unlabeled sample.

OpenMS contains tools that analyze data from SILAC experiments.

Video

For more information on SILAC, view the following links:

Label-free quantification (LFQ)

LFQ is a cheap and natural method of quantifying molecules of interest. As the name suggests, no labeling of molecules is involved.

LFQ includes the following steps:

1. **Conduct replicate experiments.**
2. **Generate LC-MS maps** for each experiment.
3. **Find features** in all LC-MS maps. A *feature* is a collection of peaks that belong to a chemical compound.
4. **Align maps** to address shifts in retention times.
5. **Match corresponding features** in different maps. We refer to this as **grouping** or **linking**.
6. **Identify feature groups**, called *consensus features*.
7. **Quantify consensus features.**

Video

For more information on LFQ, [view this video](#). For more information on the steps involved in LFQ, [view this video](#)

Feature finding

Feature finding is method for identifying all peaks belonging to a chemical compound. Feature finding involves the following steps:

1. **Extension** where we collect all data points we think belong to the peptide.
2. **Refinement** where we remove peaks that we think do not belong to the peptide.
3. **Fit an optimal model** to the isolated peaks.

The above steps are iterative; we repeat these steps until no improvement can be made to the model.

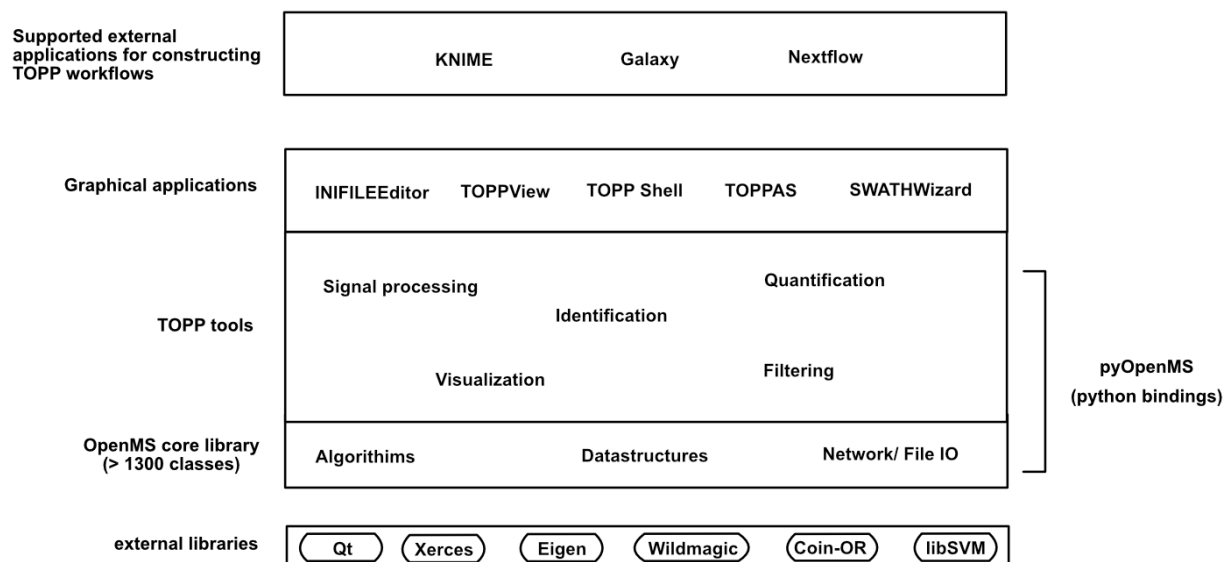
OpenMS contains a number of feature finding algorithms.

Video

For more information on feature finding, [view this video](#).

1.3 Entry Points to OpenMS

OpenMS has been structured so that users from a wide range of fields can access what they need to solve their particular problem, depending on their skillset.



The following entry points for OpenMS and its TOPP tools are available for users (click the card for more information):

Workflows Use a supported workflow editor to create or run predefined workflows. Suppose you want to run the same sequence of TOPP tools on a number of data sets. You can use applications such as KNIME, Nextflow and Galaxy (where TOPP tools are available as a plugin), to apply predefined workflows or custom workflows you have designed on your data.

Graphical apps Use OpenMS graphical user interfaces to easily process data and inspect results. When OpenMS is installed, a number of graphical user interfaces are available. Life science experts that want to quickly process their mass spectrometry data with the TOPP tools available can use this option.

Command-line tools Use over 100 command-line tools to automate pre-defined tasks efficiently. All TOPP tools can be executed from a Command Line Interface (CLI) directly or using a shell script. By using a CLI, users can easily automate tasks and create workflows that can be saved, stored and used on multiple datasets. Command line interfaces include, but are not limited to PowerShell in Windows or Terminal in Linux or macOS.

pyOpenMS Use the pyOpenMS python library to rapidly prototype methods and scripts. Classes and methods originally written in C++ have been exposed to a Python interface (pyOpenMS) using Python bindings (via Cython). Central data structures even provide fast export to pandas dataframes or numpy arrays. pyOpenMS was created for users with Python knowledge who want to quickly prototype new methods and scripts or interface with other prominent data science, machine learning or visualization libraries such as tensorflow or plotly.

OpenMS C++ core library Build the OpenMS C++ core library from source to develop your own efficient tools and methods. As shown in the image above, TOPP tools have been created using the OpenMS core library and some external libraries, which are written in C++. Using the OpenMS core library directly provides faster access to tools and shorter run-times. Additional TOPP tools can also be developed, customized or extended based on the user's needs.

1.4 Installation

1.4.1 GNU/Linux

Install via Conda

Warning: At this time, we do not provide a conda package for our GUI tools. This means if you want to install e.g., TOPPView or SwathWizard for use in for example one of our tutorials, please refer to a different installation method below.

You can use conda or mamba to install the OpenMS library and tools without user interface. Depending on the conda channel, you can obtain release versions (bioconda channel) and nightly versions (openms channel).

1. Follow the instructions to install [conda](#) or [mamba](#). In the following, every mention of conda may be substituted by mamba for faster environment solving.
2. We recommend to create a new environment with one of the supported python version versions:

```
conda create -n openms python=3.10
```

3. Add some channels to find dependencies:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Note: You can also add the channels for your current environment only with the `--env` option.

Warning: The order of the channels is important!

Note: conda-forge might already be added if you are using Mambaforge.

4. Install any of the following packages related to OpenMS

openms

openms contains all OpenMS C++ command-line tools. GUI applications like TOPPView currently cannot be installed via conda.

libopenms

libopenms is the C++ library required for the OpenMS C++ Tools to work. This is also an auto-installed dependency of openms.

pyopenms

pyopenms is the python package that allows to use algorithms from libopenms in Python.

openms-thirdparty

openms-thirdparty are external tools that are wrapped in OpenMS with adapters. This package is required to use the adapters in the openms package.

Warning: Due to unavailability of a large part of the thirdparty tools for macOS via conda, we are not providing a openms-thirdparty package on macOS either.

via bioconda for release versions

openms

```
conda install openms
```

libopenms

```
conda install libopenms
```

pyopenms

```
conda install pyopenms
```

openms-thirdparty

```
conda install openms-thirdparty
```

or our own openms channel for nightly snapshots (which are build based on the same bioconda dependencies)

openms

```
conda install -c openms openms
```

libopenms

```
conda install -c openms libopenms
```

pyopenms

```
conda install -c openms pyopenms
```

openms-thirdparty

```
conda install -c openms openms-thirdparty
```

Install via package managers

Packaged versions of **OpenMS** are provided for Fedora, OpenSUSE, Debian, and Ubuntu. You can find them to download [here](#). For other GNU/Linux distributions or to obtain the most recent version of the library, installation should be done via building from the source code.

Important: These packages are not directly maintained by the OpenMS team and they can not be guaranteed to have the same behaviour as when building it from source code. Also, their availability and version is subject to change and support might be limited (due to unforeseen or untested behaviour). It is suggested not to install them parallel to our Debian package.

Note: Some thirdparty software used via adapter tools in OpenMS might also require an installed JavaVM.

Install via the provided Debian package

For Debian-based Linux users, it is suggested to use the [deb-package](#) provided. It is most easily installed with **gdebi** which automatically resolves the dependencies available in the PPA Repositories.

```
sudo apt-get install gdebi
sudo gdebi /PATH/TO/OpenMS.deb
```

If you encounter errors with unavailable packages, troubleshoot using the following steps.

1. Qt5 (or one of its packages, e.g. `qt5xbase`) is missing.

It might be because your Debian is too old to have a recent enough version in its official repositories. It is suggested to use the same packages that are used while building (make sure to adapt the Qt version and your Debian/Ubuntu version, here Xenial):

```
sudo add-apt-repository ppa:beineri/opt-qt59-xenial
sudo apt-get update
```

Run the installation again.

2. ICU with its `libcicu` is missing.

You can find the missing version on [pkgs.org](#) and install it with `gdebi`, too. You can have multiple versions of ICU installed.

3. Error while executing a tool

To ensure the tool functionality, make sure you add the `OPENMS_DATA_PATH` variable to your environment as follow `export OPENMS_DATA_PATH=/usr/share/OpenMS`

4. Thirdparty installation of Qt5 in step 1

Make sure you source the provided environment file using: `source /opt/qt59/bin/qt59-env.sh`

5. Adapters are not finding thirdparty applications

Executables for thirdparty applications can be found in: `/usr/share/OpenMS/THIRDPARTY` Add the folders in your `PATH` for a convenient use of the adapters.

Run OpenMS inside a (Bio)Container

1. Install a containerization software (e.g., [Docker](#) or [Singularity](#))

2. Pull an image from one of the following registries:

- [OpenMS GitHub Container Registry](#) for nightly binaries AND releases:

On our registry, we provide one image for the library (with contrib) and one for the executables (with thirdparty).

1. `openms-library`
2. `openms-executables`

They can be pulled/run via the following commands:

Docker

```
docker pull ghcr.io/openms/openms-library
docker pull ghcr.io/openms/openms-executables
```

Singularity

```
singularity run ghcr.io/openms/openms-library-sif
singularity run ghcr.io/openms/openms-executables-sif
```

Note: Per default this results in the download of the latest nightly snapshot. Specify a release version (e.g., `docker pull ghcr.io/openms/openms-library:3.0.0`) to receive a stable version.

- Otherwise, the [BioContainers Registries](#) and the associated Galaxy project provide native containers based on our bioconda packages for both Docker and Singularity.
 1. [BioContainers libopenms](#)
 2. [BioContainers openms](#)
 3. [BioContainers openms-thirdparty](#)
 4. [BioContainers pyOpenMS](#)

Images of the containers can be pulled via or one of the following commands:

Docker

```
docker pull quay.io/biocontainers/libopenms
docker pull quay.io/biocontainers/openms
docker pull quay.io/biocontainers/pyopenms
docker pull quay.io/biocontainers/openms-thirdparty
```

Singularity

```
singularity run https://depot.galaxyproject.org/singularity/libopenms
singularity run https://depot.galaxyproject.org/singularity/openms
singularity run https://depot.galaxyproject.org/singularity/pyopenms
singularity run https://depot.galaxyproject.org/singularity/openms-thirdparty
```

Note: If Singularity images fail to download or run, try to use the Docker images as Singularity will automatically convert them.

Dockerfiles to build different kind of images (e.g., for ArchLinux) yourself can be found on GitHub in our [OpenMS/dockerfiles](#) repository. They usually follow our build instructions closely, so you can have a look on how this is done in a clean environment.

Build OpenMS from source

To build OpenMS from source, follow the build instructions for [Linux](#).

1.4.2 macOS

Install via Conda

Warning: At this time, we do not provide a conda package for our GUI tools. This means if you want to install e.g., TOPPView or SwathWizard for use in for example one of our tutorials, please refer to a different installation method below.

You can use conda or mamba to install the OpenMS library and tools without user interface. Depending on the conda channel, you can obtain release versions (bioconda channel) and nightly versions (openms channel).

1. Follow the instructions to install [conda](#) or [mamba](#). In the following, every mention of conda may be substituted by mamba for faster environment solving.
2. We recommend to create a new environment with one of the supported python version versions:

```
conda create -n openms python=3.10
```

3. Add some channels to find dependencies:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Note: You can also add the channels for your current environment only with the `--env` option.

Warning: The order of the channels is important!

Note: conda-forge might already be added if you are using Mambaforge.

4. Install any of the following packages related to OpenMS

openms

openms contains all OpenMS C++ command-line tools. GUI applications like TOPPView currently cannot be installed via conda.

libopenms

libopenms is the C++ library required for the OpenMS C++ Tools to work. This is also an auto-installed dependency of openms.

pyopenms

pyopenms is the python package that allows to use algorithms from libopenms in Python.

openms-thirdparty

openms-thirdparty are external tools that are wrapped in OpenMS with adapters. This package is required to use the adapters in the openms package.

Warning: Due to unavailability of a large part of the thirdparty tools for macOS via conda, we are not providing a openms-thirdparty package on macOS either.

via bioconda for release versions

openms

```
conda install openms
```

libopenms

```
conda install libopenms
```

pyopenms

```
conda install pyopenms
```

openms-thirdparty

```
conda install openms-thirdparty
```

or our own openms channel for nightly snapshots (which are build based on the same bioconda dependencies)

openms

```
conda install -c openms openms
```

libopenms

```
conda install -c openms libopenms
```

pyopenms

```
conda install -c openms pyopenms
```

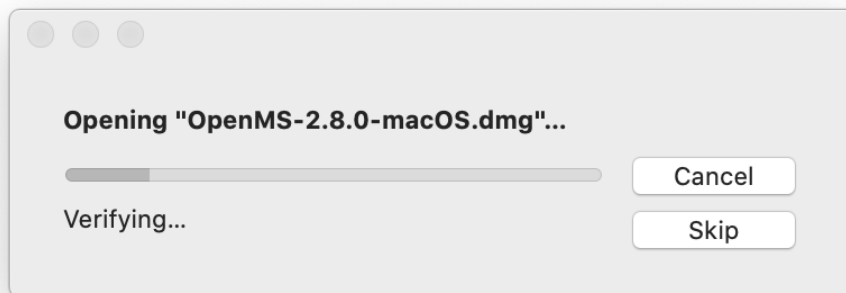
openms-thirdparty

```
conda install -c openms openms-thirdparty
```

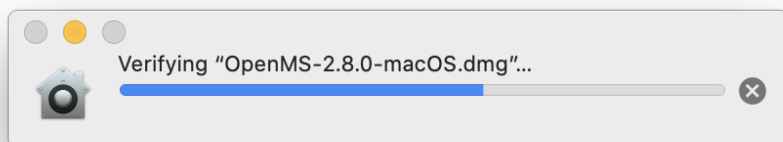
Install via macOS installer

To install OpenMS on macOS, run the following steps:

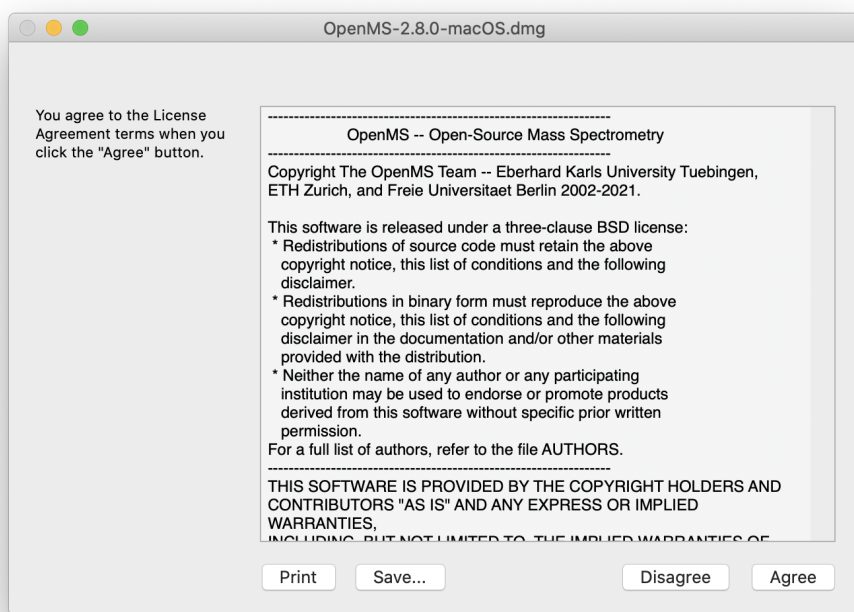
1. Download and install the macOS drag-and-drop installer from the [archive](#).
2. Double click on the downloaded file. It will start to open the OpenMS-<version>-macOS.dmg disk image file.



3. Verify the download.



4. Agree to the license agreements.



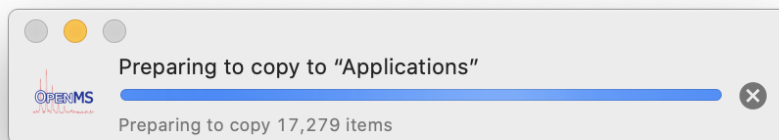
5. Drag OpenMS to the Applications folder.

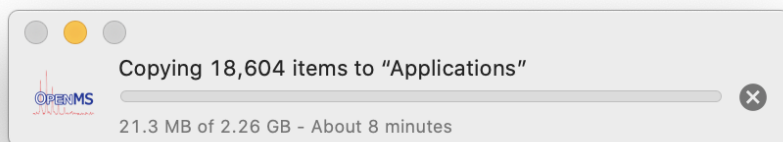


To install, please drag
the OpenMS folder to
your Applications folder



6. It will start copying to applications.





To use *TOPP* as regular app in the shell, add the following lines to the `~/.profile` file.

Warning: Known Installer Issues

1. Nothing happens when you click OpenMS apps or the validity of the developer could not be confirmed.

This usually means the OpenMS software lands in quarantine after installation of the `.dmg`. Since macOS Catalina (maybe also Mojave) all apps and executables have to be officially notarized by Apple but we currently do not have the resources for a streamlined notarization workflow.

To have a streamlined experience without blocking popups, it is recommended to remove the quarantine flag manually, using the following steps:

Open the Terminal.app and type the following (replace the first line with the actual installation directory):

```
cd /Applications/OpenMS-<version>
sudo xattr -r -d com.apple.quarantine *
```

2. Bug with running Java based thirdparty tools like *MSGFPlusAdapter* and *LuciphorAdapter* from within **TOPPAS.app**

If you face issues while running Java based thirdparty tools from within *TOPPAS.app*, run the *TOPPAS.app* from within the Terminal.app (e.g. with the `open` command) to get access to the path where Java is located. Java is usually present in the `PATH` of the terminal. Advanced users can set this path in the `Info.plist` of/inside the *TOPPAS.app*.

```
export OPENMS_TOPP_PATH=<OpenMS-PATH>
source ${OPENMS_TOPP_PATH}/.TOPP_bash_profile
```

Make sure `<OpenMS-PATH>` points to the folder where OpenMS is installed locally (e.g., `/Applications/OpenMS-<version>`)

Run OpenMS inside a (Bio)Container

1. Install a containerization software (e.g., [Docker](#) or [Singularity](#))
2. Pull an image from one of the following registries:
 - [OpenMS GitHub Container Registry](#) for nightly binaries AND releases:

On our registry, we provide one image for the library (with contrib) and one for the executables (with thirdparty).

1. `openms-library`
2. `openms-executables`

They can be pulled/run via the following commands:

Docker

```
docker pull ghcr.io/openms/openms-library
docker pull ghcr.io/openms/openms-executables
```

Singularity

```
singularity run ghcr.io/openms/openms-library-sif
singularity run ghcr.io/openms/openms-executables-sif
```

Note: Per default this results in the download of the latest nightly snapshot. Specify a release version (e.g., `docker pull ghcr.io/openms/openms-library:3.0.0`) to receive a stable version.

- Otherwise, the [BioContainers Registries](#) and the associated Galaxy project provide native containers based on our bioconda packages for both Docker and Singularity.
 1. [BioContainers libopenms](#)
 2. [BioContainers openms](#)
 3. [BioContainers openms-thirdparty](#)
 4. [BioContainers pyOpenMS](#)

Images of the containers can be pulled via or one of the following commands:

Docker

```
docker pull quay.io/biocontainers/libopenms
docker pull quay.io/biocontainers/openms
docker pull quay.io/biocontainers/pyopenms
docker pull quay.io/biocontainers/openms-thirdparty
```

Singularity

```
singularity run https://depot.galaxyproject.org/singularity/libopenms
singularity run https://depot.galaxyproject.org/singularity/openms
singularity run https://depot.galaxyproject.org/singularity/pyopenms
singularity run https://depot.galaxyproject.org/singularity/openms-thirdparty
```

Note: If Singularity images fail to download or run, try to use the Docker images as Singularity will automatically convert them.

Dockerfiles to build different kind of images (e.g., for ArchLinux) yourself can be found on GitHub in our [OpenMS/dockerfiles](#) repository. They usually follow our build instructions closely, so you can have a look on how this is done in a clean environment.

Build OpenMS from source

To build OpenMS from source, follow the build instructions for [macOS](#).

1.4.3 Windows

Install via Windows installer

To Install the binary package of OpenMS & *TOPP*:

1. Download and execute the installer `OpenMS-<version>-Win64.exe` from the [archive](#) and follow its instructions.
2. Run the installer under the user account that later runs OpenMS. It is not advised to use admin account for installation. When asked for an admin authentication, please enter the credentials.

Tip: The windows binary version works with most versions of windows from Win7 to Win10 (older versions might still work but are untested).

Known issues

1. During installation, an error message pops up, saying:

“The installation of the Microsoft .NET 3.5 SP1’ package failed!

You must download and install it manually in order for Proteowizard to work. This should only happen if installation is done by selecting the “Third Party - Proteowizard” components. The reason is usually that **.NET 3.5 SP1** is already installed (see Windows Control Panel). If it’s not installed, follow the instructions of the error message.

2. During installation, an error message pops up, saying:

“The installation of the Visual Studio redistributable package ... failed. ...”

This is a known issue with a Microsoft package, we cannot do anything about it. The error message will give the location where the redistributable package was extracted to. Go to this folder and run the executable (usually named `vcredistXXXX.exe`) as an administrator (right-click and then select **Run-As**). You will likely receive an error message (this is also the reason why the OpenMS setup complained about it). You might have to find the solution to fix the problem in your local machine. If you’re lucky the error message is instructive and the problem is easy to fix.

3. During installation, an error message pops up saying:

“Error opening installation log file”

To fix, check the system environment variables. Make sure they are apt. There should a `TMP` and a `TEMP` variable, and both should contain one directory only, which exists and is writable. Fix accordingly (search the internet on how to change environment variables on Windows).

4. For Win8 or later, Windows will report an error while installing `.net4` as it’s mostly included. But it might occur that `.net3.5` does not get properly installed during the process.

Fix is to enable the .NET Framework 3.5 yourself through Control Panel. See this [Microsoft help page.aspx#ControlPanel](#)) for detailed information. Even if this step fails, this does not affect the functionality of OpenMS, except for the executability of included third party tools (ProteoWizard).

For instructions on how to install the OpenMS graphical and command-line tools, choose your operating system from the items below.

GNU/Linux [Installation on Linux](#)

macOS [Installation on macOS](#)

Windows [Installation on Windows](#)

1.5 OpenMS Graphical User Interfaces

OpenMS provides a suite of graphical user interfaces, designed for users who want easy access to TOPP tools. These interfaces include:

- **INIFileEditor**

A GUI application used to edit TOPP INI files. TOPP INI files are used to configure TOPP tool parameters. TOPP INI files are files with the extension `.ini`. For more information, read our [INIFile Editor](#) section.

- **TOPPView**

A GUI application used to inspect, visualize and compare mass spectrometry data. For more information, read our [TOPPView: Visualize with OpenMS](#) section.

- **TOPPAS (deprecated)**

A GUI application used to apply multiple tools sequentially on mass spectrometry data. Applying multiple tools in a sequence is referred to as a workflow or a pipeline. OpenMS no longer supports TOPPAS and instead recommends the use of [KNIME](#), for which we provide a community plugin.

- **SwathWizard** An application for SWATH analysis. SwathWizard is used to analyze DIA swath data. For more information, read our [SwathWizard](#) section.

A possible workflow would consist of the following steps:

1. Generate a TOPP INI file from the [command line](#).
2. Edit the TOPP INI file in the INIFile Editor.
3. Import data into TOPPView.
4. Apply TOPP tool to data in TOPPView. You will need to load the TOPP INI file edited in step 1.

1.6 TOPP Tools

The following sections introduce TOPP, some general concepts and more specific information regarding each tool.

1.6.1 Introduction to TOPP

TOPP - The OpenMS Pipeline is a set of tools for the analysis of HPLC-MS data. These tools can be either:

- *Executed from the command line* or,
- Applied individually using OpenMS graphical applications.
- Applied in sequence as a workflow using a workflow editor such as KNIME, Nextflow or Galaxy.

Before you choose one of the above options, there are few concepts that need to be understood.

File formats

OpenMS only accepts files in certain formats, including but not limited to:

- **mzML**: The HUPO-PSI standard format for mass spectrometry data.
- **featureXML**: The OpenMS format for quantitation results.
- **consensusXML**: The OpenMS format for grouping features in one map or across several maps.
- **idXML**: The OpenMS format for protein and peptide identification.

Documented schemas of the OpenMS formats can be found [here](#).

If your data is not in the above formats, you may need to use a file conversion TOPP tool.

TOPP INI files

TOPP INI files are XML-based files with an `.ini` extension. OpenMS uses TOPP INI files to set parameters for one or more TOPP tools. Alternatively, the command line can be used to set TOPP tool parameters. Here is an example of a TOPP INI file:

```
<PARAMETERS>

<NODE name="FileFilter">
  <NODE name="1">
    <ITEM name="rt" value="0:1200" type="string"/>
  </NODE>
  <NODE name="2">
    <ITEM name="mz" value="700:1000" type="string"/>
  </NODE>
</NODE>

<NODE name="common">
  <NODE name="FileFilter">
    <ITEM name="rt" value=":" type="string"/>
    <ITEM name="mz" value=":" type="string"/>
  </NODE>
  <ITEM name="debug" value="2" type="int"/>
</NODE>

</PARAMETERS>
```

Features, feature maps and featureXML files

An LC-MS feature is a construct in OpenMS that is used to describe a 2D peak caused by an analyte interacting with the stationary phase. Each feature contains the following metadata: an id, retention time, mass-to-charge ratio, intensity, overall quality and one or more convex hulls.

A feature map is a container for features. One feature map can contain many features.

A featureXML file is an XML based file which contains one feature map.

FeatureXML files can be created from mzML files using OpenMS's feature detection algorithms.

Consensus feature, consensus maps, consensusXML files

A consensus feature is a special type of LC-MS feature that is quantified across multiple experiments. A consensus feature is formed by linking or grouping features with similar mass-to-charge ratios and intensities from various experiment runs. Each consensus feature references the features used to form the consensus feature.

Similar to a feature map, a consensus map is a container for consensus features. One consensus map can contain many consensus features.

ConsensusXML files can be created from featureXML files using OpenMS's feature grouping algorithms.

1.6.2 Types of TOPP Tools

The following tools are offered:

- **File conversion**

TOPP file conversion tools can be used to convert files into a supported format.

- **File handling**

TOPP file handling tools are largely used to extract or merge data. For more information, view the File handling.

- **Centroiding**

The conversion of the “raw” ion count data acquired by the machine into peak lists for further processing is usually called peak picking or centroiding. The choice of the algorithm should mainly depend on the resolution of the data. OpenMS provides different algorithms for centroiding depending on the resolution of the data. For more information, view the Picking peaks section.

- **Spectrum processing**

A number of spectrum processing tools are available. These include peak filtering and peak normalization tools, as well as other miscellaneous tools.

- **Mass correction and calibration**

To ensure that your data is sound, OpenMS have provided a number of mass correction and calibration tools. The types of tools used will depend on the type of equipment you have employed. For more information, view the Calibration section.

- **Spectrum clustering**

Spectrum clustering is the grouping of spectra that have many peaks in common. OpenMS provides tools for spectrum clustering to identify molecules in large datasets more efficiently.

- **Map alignment**

When looking to identify molecules, it is common to run multiple experiments, where each experiment produces a set of data. In OpenMS, every set of data is represented by a feature map. Before combining feature maps

to create a consensus map, it is advised to use OpenMS's map alignment tools so that all your datasets are comparable and based on a common retention time axis. For more information, view the Map alignment section.

- **Feature linking**

OpenMS provides a number of algorithms for feature grouping or linking. For more information, view the Feature grouping section.

- **Quantitation**

A number of tools are available that allow for the identification and quantification of features. The tools you use will depend on the type of mass spectrometry experiment you have set up, and the type of molecules you wish to identify. For more information, view the Feature detection and Feature detection on centroided data sections.

- **Protein/Peptide identification**

- **Protein/Peptide processing**

- **Targeted experiments and OpenSWATH**

- **Peptide property prediction**

For more information, view the Peptide property prediction section.

- **Cross-linking**

Cross-linking is a technique where substances are chemically treated to create covalent bonds between different molecules. The strength of the covalent bonds can be quantified to indicate the proximity of certain molecules within a 3D structure.

- **Quality control**

OpenMS provides tools to measure the quality of LC-MS data. For more information, view the Quality control section.

For the full list of TOPP tools, visit the [API reference](#) website.

1.7 UTILS Tools

There are also a number of tools in the beta stage called UTILS. They include but are not limited to:

- Signal processing and preprocessing
- File handling
- Algorithm evaluation
- Protein/peptide identification
- Cross-linking
- Quantitation
- Metabolite identification
- Targeted experiments and OpenSWATH
- RNA
- Quality control

Additional information for the ProteomicsLFQ Quantitation UTILS tool is provided. For the full list of UTILS tools, visit the [API Reference](#) website.

1.8 Command Line Interface

TOPP tools are designed to be called from the command line. OpenMS provides a Command Line Interface (CLI) called TOPP shell to easily execute TOPP tools on mass spectrometry data. However, you can configure the CLI of your choice to run TOPP tools.

Command line calls will depend on the TOPP tools used, as each TOPP tool has its own set of parameters. However, the following arguments are typically used:

- **-in**

Specify an input file in the command line using the **-in** argument. The input file should be in a supported format. If not, use the file converter to convert the file to one of the supported formats. For more information, view the file handling documentation.

- **-out**

Specify an output file in the command line using the **-out** argument.

- **-ini**

Specify an INI file in the command line using the **-ini** argument. TOPP uses INI files to set parameters specific to the command line tool being called.

- **-write_ini**

Create an INI file using the **-write_ini** file argument. Create an INI file with this call: `<insert TOPP tool> -write_ini <insert output INI File>` If you want a visual tool to assist setting parameters, use INIFileEditor, an application provided when you download OpenMS. Otherwise, you can set the parameters from the command line.

- **-help**

Get information about basic options related to the tool using the **-help** parameter. For more advanced options (algorithmic parameters), use **--help**.

- **--help**

Get detailed information about algorithmic parameters using the **--help** parameter.

Many (but not all) command line calls will have the following structure:

```
<insert TOPP tool> -in <insert input mzML file> -out <insert output mzML file> -ini  
↪<insert INI file>
```

The following command line call uses the FileFilter tool to extract data from an mzML file. Note, that this call directly specifies the tool-specific parameters and doesn't rely on an INI file:

FileFilter	-in myinfile.mzML	-levels 2 -rt 100:1500	-out myoutfile.mzML
└──────────┘	└──────────────────┘	└──────────────────────────┘	└──────────────────┘
TOPP tool	input file	tool-specific parameters	output file

1.9 TOPPView: Visualize with OpenMS

1.9.1 Introduction

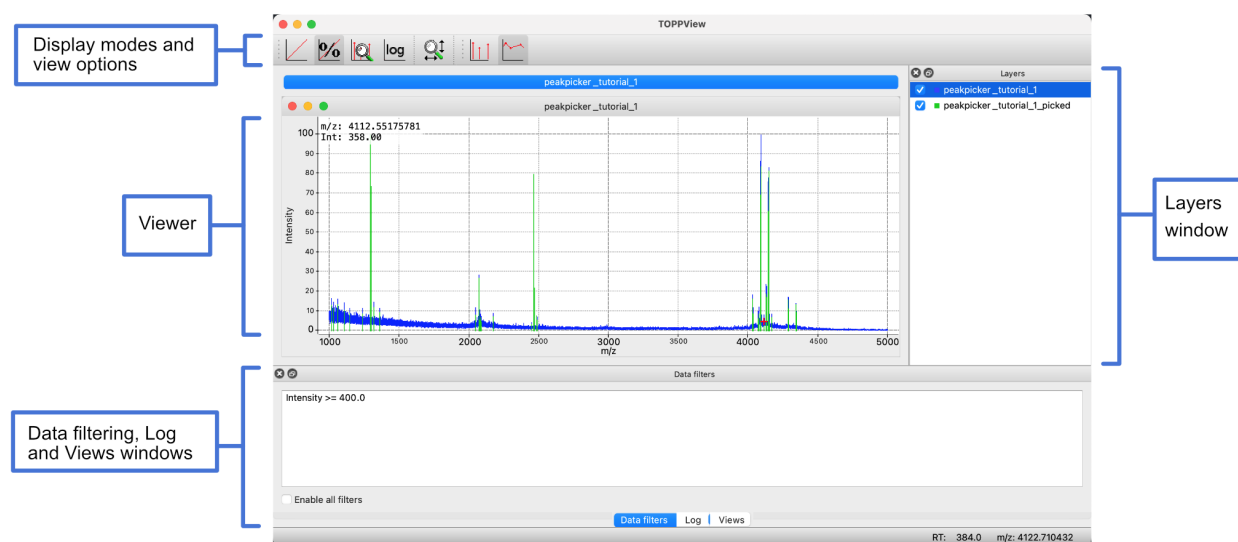
TOPPView is a graphical application for inspecting and visualizing *MS* and *HPLC-MS* data. It can be used to inspect files in *mzML*, *mzData*, *mzXML* and several other text-based file formats.

In each view, several datasets can be displayed simultaneously using the layer concept. This allows visual comparison of several datasets as well as displaying input data and output data of an algorithm together.

TOPPView is intended for visual inspection of the data by experimentalists as well as for analysis software by developers.

1.9.2 User interface

The following image illustrates different components of TOPPView's user interface.



Components include:

- **Display modes and options**
- A **Viewer** that displays visual data.
- **Layers**

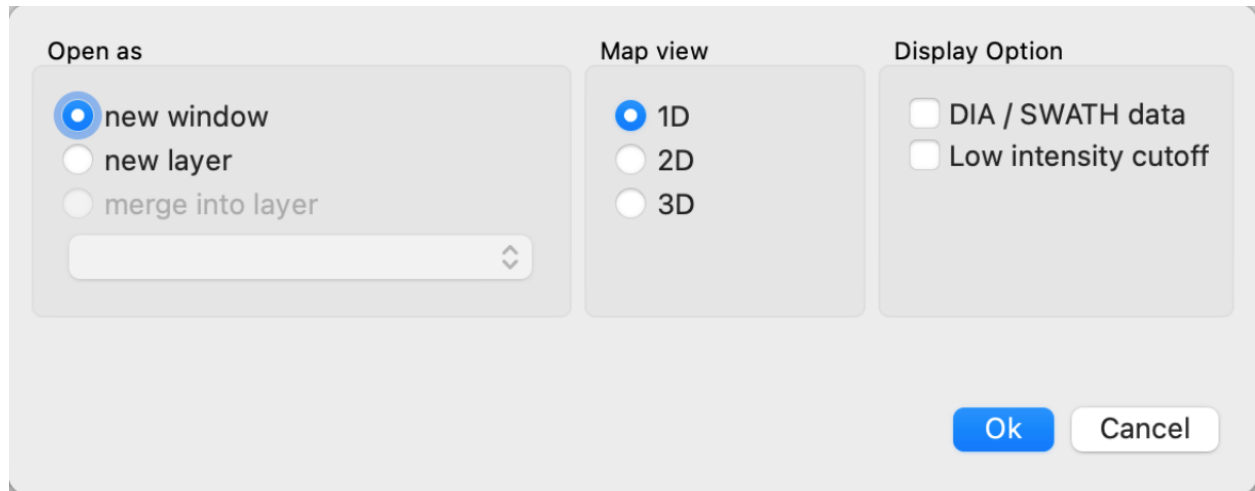
You can visualize several datasets by creating **layers**. In the **Layers window** (dock window in the upper right corner), layers can be hidden and shown using the check box in front of each layer name.

- **Data filtering window**
- **Log window** to view textual output of applying TOPP tools to data.
- **Views windows** that show tabulated information about the dataset.

1.9.3 Import a file

To import data into TOPPView:

1. Go to **File > Open file**.
2. Choose a file from the file importer and click **Open**.
3. Select options from the following panel and click **Ok**:



You can choose to open the new dataset as a **new window** or **new layer**. Choosing **new window** will open a new tab. If you are planning on comparing multiple datasets and want to view them all at once, choose **new layer**.

TOPPView automatically selects the **Map view** depending on the data you have imported.

You should be able to see your data in the **Viewer**.

1.9.4 Process data in TOPPView

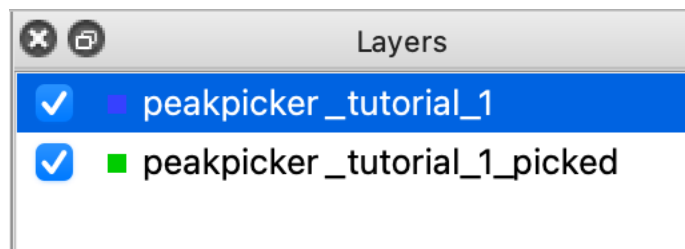
The following section details how to apply TOPP tools and filter data in TOPPView.

Apply TOPP tool

OpenMS provides a number of TOPP tools that can be applied to your data.

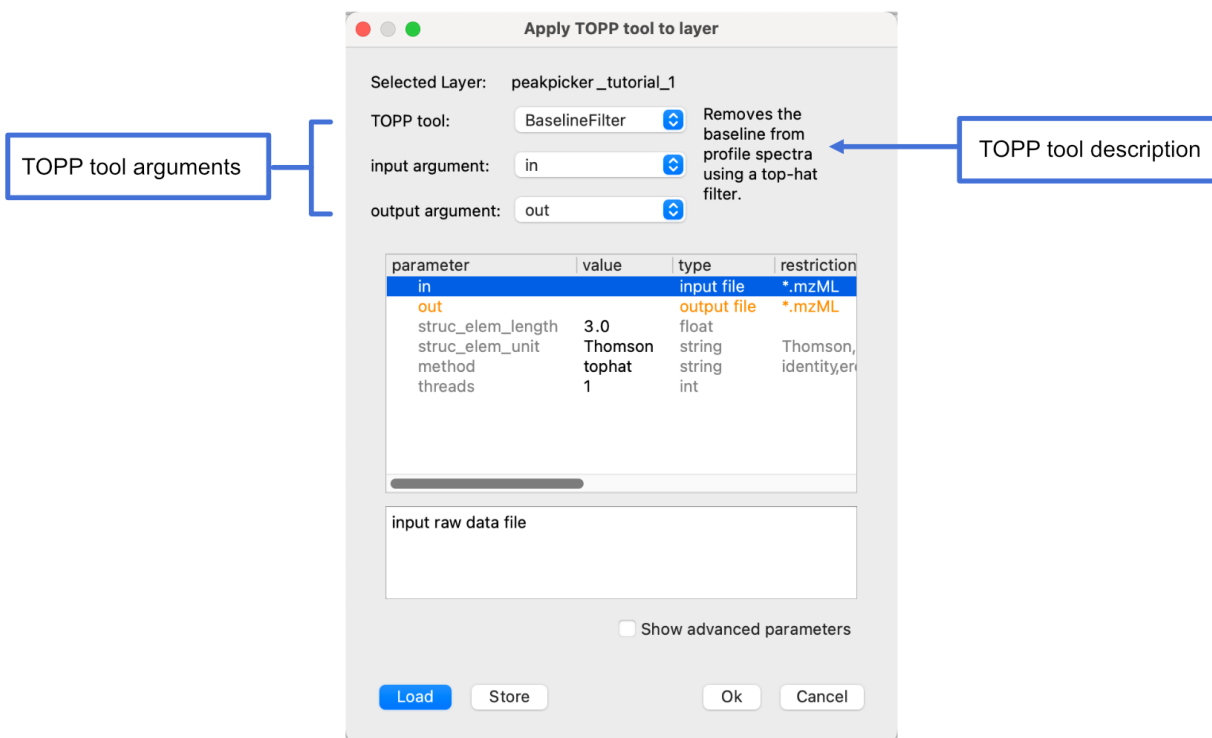
To apply a TOPP tool to your dataset:

1. Select a layer in the **Layers window**. The selected layer will be highlighted blue.

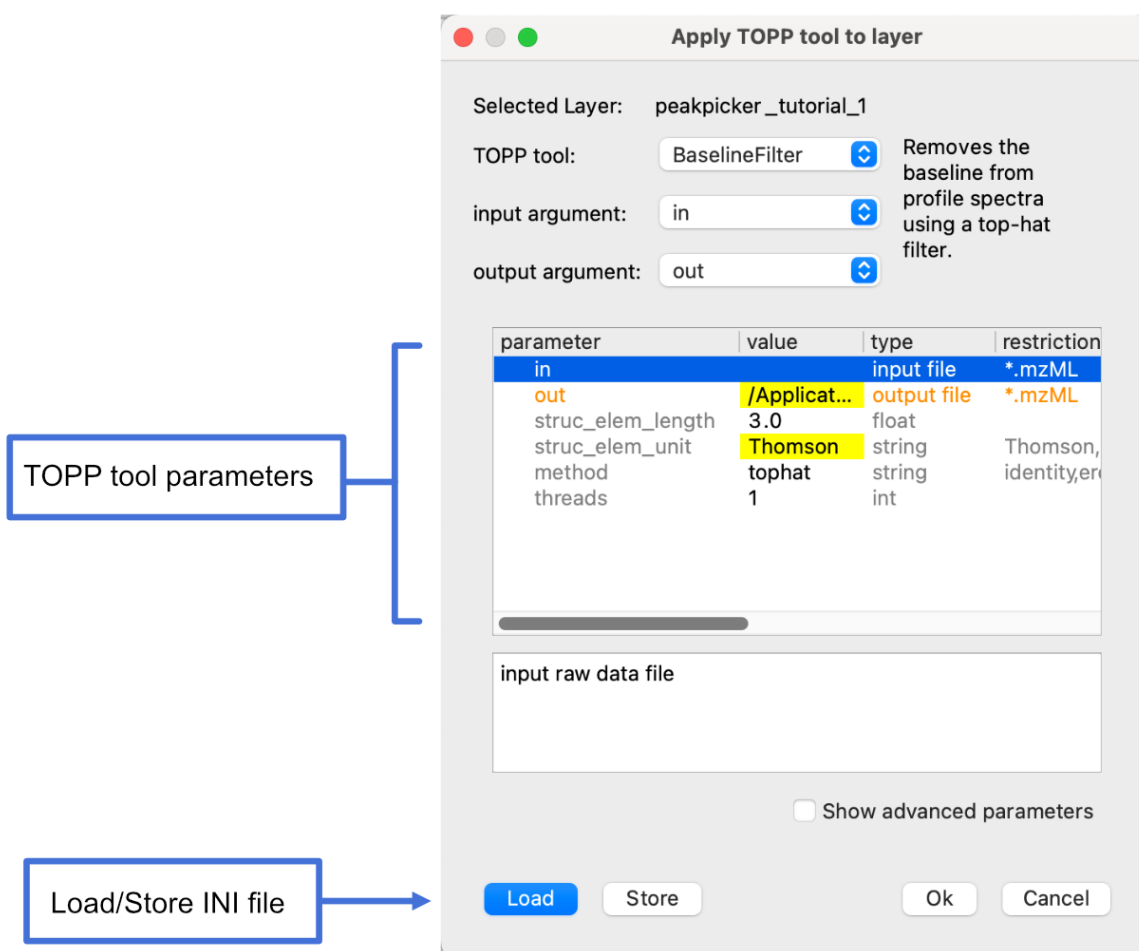


2. Go to **Tools > Apply TOPP tool to whole layer**. This will open a panel to select and configure your TOPP tool.

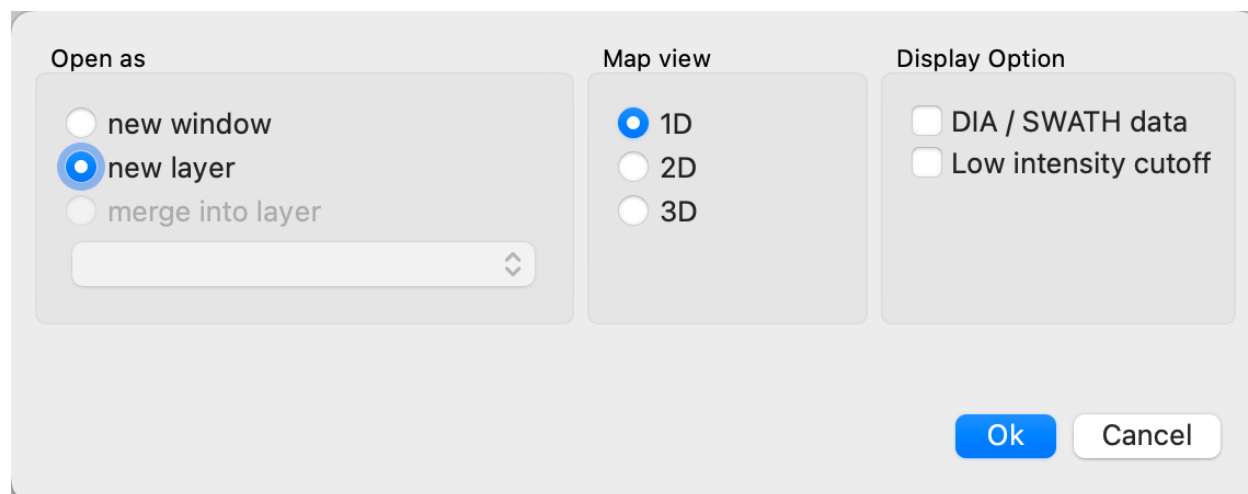
3. Select a **TOPP tool** from the dropdown menu. A description of the TOPP tool will be displayed on the right. You may have to also specify the **input argument** as well though TOPPView might automatically select this option for you. To save the output to a file, specify the **output argument**.



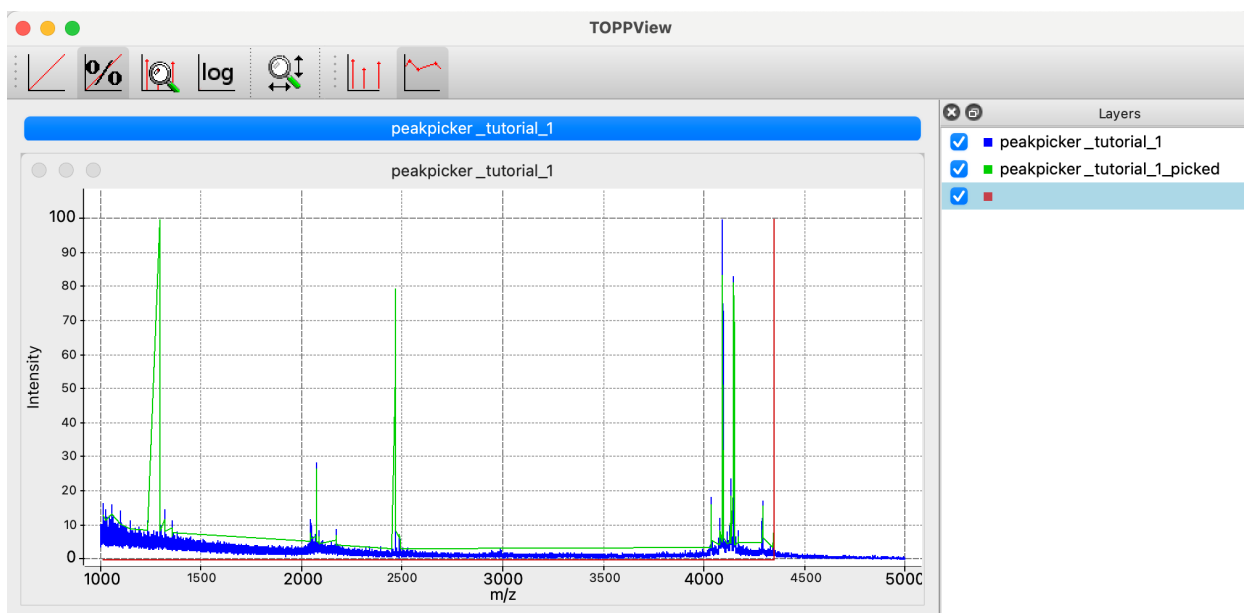
4. Specify the TOPP tool parameters by either:



5. Click **Ok**. You will be prompted to load the new dataset as a **new window** or a **new layer**. Choose an option and click **Ok**.



6. If you chose to load the data in a new window, a new tab will appear. To view that data, select the tab. If you chose to load the data as a new layer, the data will be visualized in the **Viewer**. You can also see the new layer without a name in the **Layers window**.



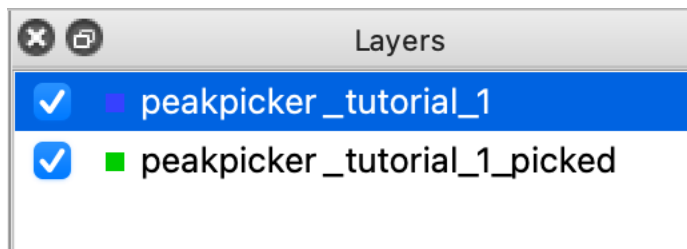
7. (Optional) If you did choose to import the data as a new layer, give the new layer a name. To do this, right-click the layer in the **Layers window** and select **Rename**. Enter a name and click OK.

Filter data

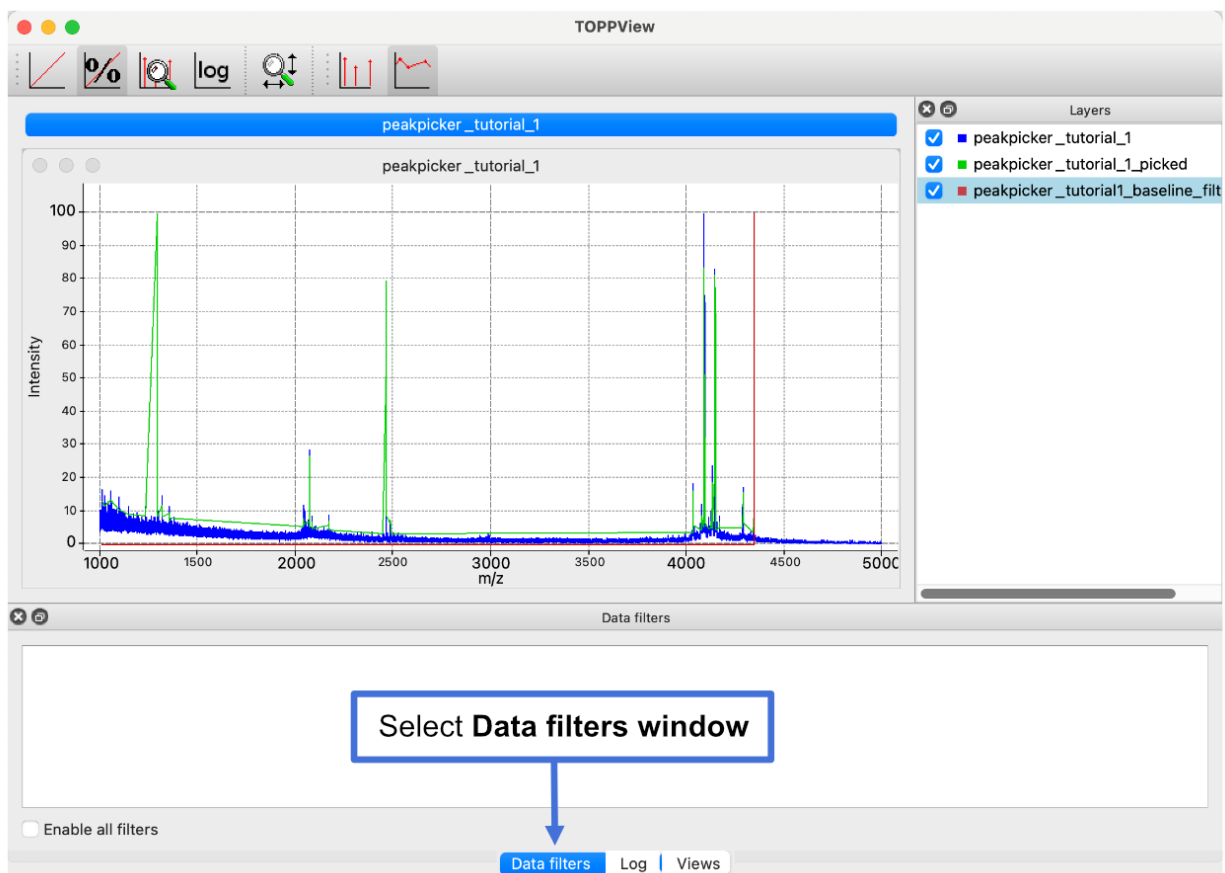
You may only want to see some data from your dataset and hide the rest. OpenMS allows you to filter data based on the following fields: **Intensity**, **Quality**, **Charge**, **Size** and **Meta data**.

To filter your data:

1. Select a layer from the **Layers window**.



2. Open the **Data filters window** by clicking the tab at the bottom of the screen.



3. Add a filter to the **Data filters window** by right-clicking the window and then selecting **Add filter** from the context menu.
4. Select a field, select an operation and enter a value. For example, to exclude all peaks with an intensity of less than 6999, set **field** to **Intensity**, **operation** to **=>** and set the value to 7000. Click **Ok** on the panel to apply the changes.

The 'Data filter' dialog box is shown. It has the following fields and controls:

- Field:** A dropdown menu with 'Intensity' selected.
- Meta name:** An empty text input field.
- Operation:** A dropdown menu with '>=' selected.
- Value:** A text input field containing '7000'.
- Buttons:** 'Ok' and 'Cancel' buttons at the bottom.

5. You should see only see data that satisfies the specified criteria.

Additional topics

You might want to check out the following topics:

Views in TOPPView

TOPPView offers three types of views – a 1D view for *spectra*, a 2D view for peak maps and feature maps, and a 3D view for peak maps. All three views can be freely configured to suit the individual needs of the user.

Action Modes and Their Uses

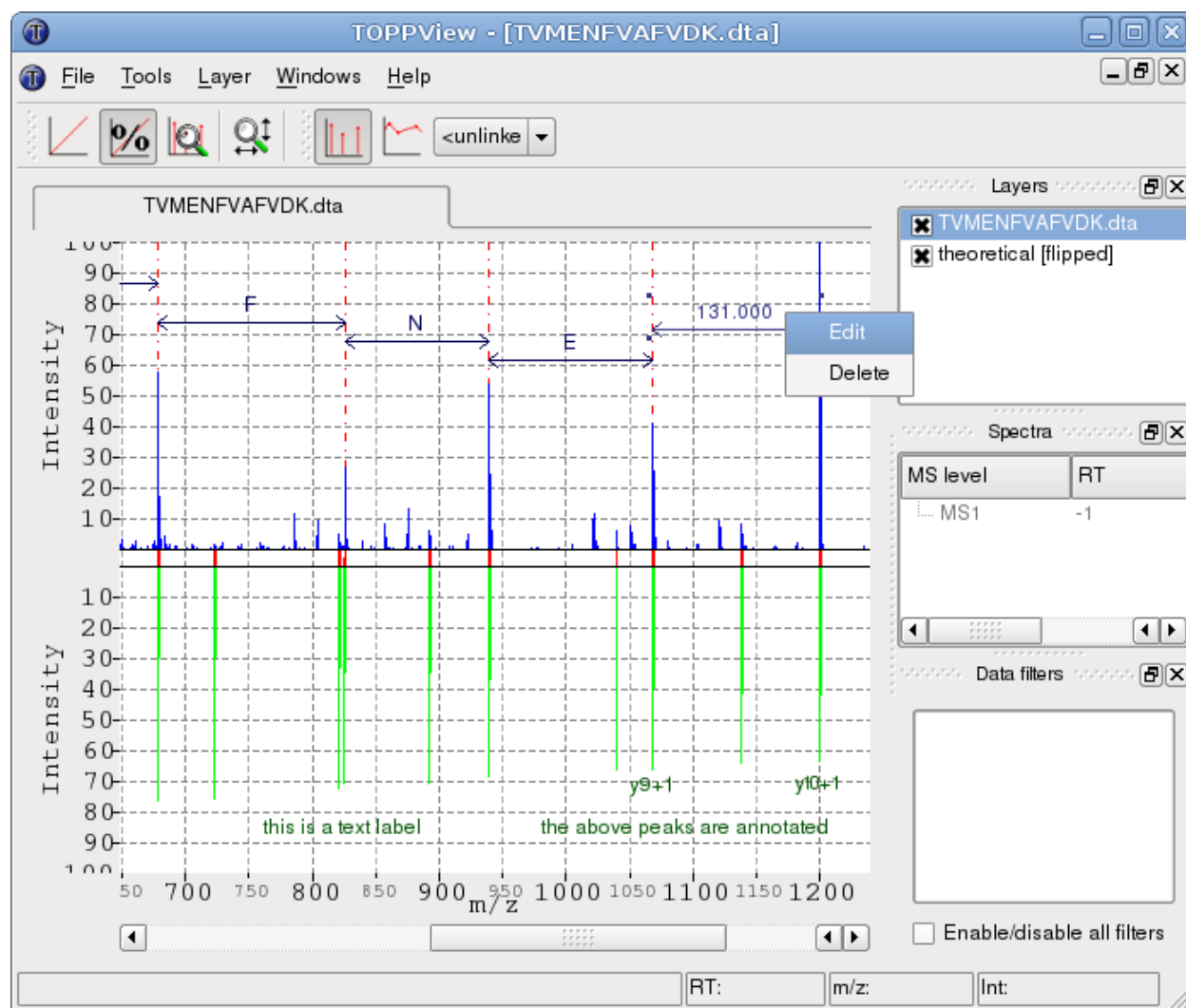
All three views share a similar interface. Three action modes are supported – one for translation, one for zooming and one for measuring:

- Translate mode
 - It is activated by default
 - Move the mouse while holding the mouse button down to translate the current view
 - Arrow keys can be used to translate the view without entering translate mode (in 1D-View you can additionally use Shift to jump to the next peak)
- Zoom mode
 - All previous zoom levels are stored in a zoom history. The zoom history can be traversed using CTRL + +/CTRL + - or the mouse wheel (scroll up and down)
 - Zooming into the data:
 - * 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.
 - * You can also use your mouse wheel to traverse the zoom history.
 - * If you have reached the end of the history, keep on pressing CTRL + + or scroll up, the current area will be enlarged by a factor of 1.25.
 - Pressing Backspace resets the zoom and zoom history.
- Measure mode
 - It is activated using SHIFT.
 - 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.

1D View

The 1D view is used to display raw spectra or peak spectra. Raw data is displayed using a continuous line. Peak data is displayed using one stick per peak. The color used for drawing the lines can be set for each layer individually. The 1D view offers a mirror mode, where the window is vertically divided in halves and individual layers can be displayed either above or below the “mirror” axis in order to facilitate quick visual comparison of spectra. When a mirror view is active, it is possible to perform a spectrum alignment of a spectrum in the upper and one in the lower half, respectively. Moreover, spectra can be annotated manually. Currently, distance annotations between peaks, peak annotations and simple text labels are provided.

The following example image shows a 1D view in mirror mode. A theoretical spectrum (lower half) has been generated using the theoretical spectrum generator (**Tools > Generate theoretical spectrum**). The mirror mode has been activated by right-clicking the layer containing the theoretical spectrum and selecting **Flip downward** from the layer context menu. A spectrum alignment between the two spectra has been performed (**Tools > Align spectra**). It is visualized by the red lines connecting aligned peaks and can be reset through the context menu. Moreover, in the example, several distances between abundant peaks have been measured and subsequently replaced by their corresponding amino acid residue code. This is done by right-clicking a distance annotation and selecting **Edit** from the context menu. Additionally, peak annotations and text labels have been added by right-clicking peaks and selecting **Add peak** annotation or by right clicking anywhere and selecting **Add Label**, respectively. Multiple annotations can be selected by holding down the CTRL key while clicking them. They can be moved around by dragging the mouse and deleted by pressing DEL.



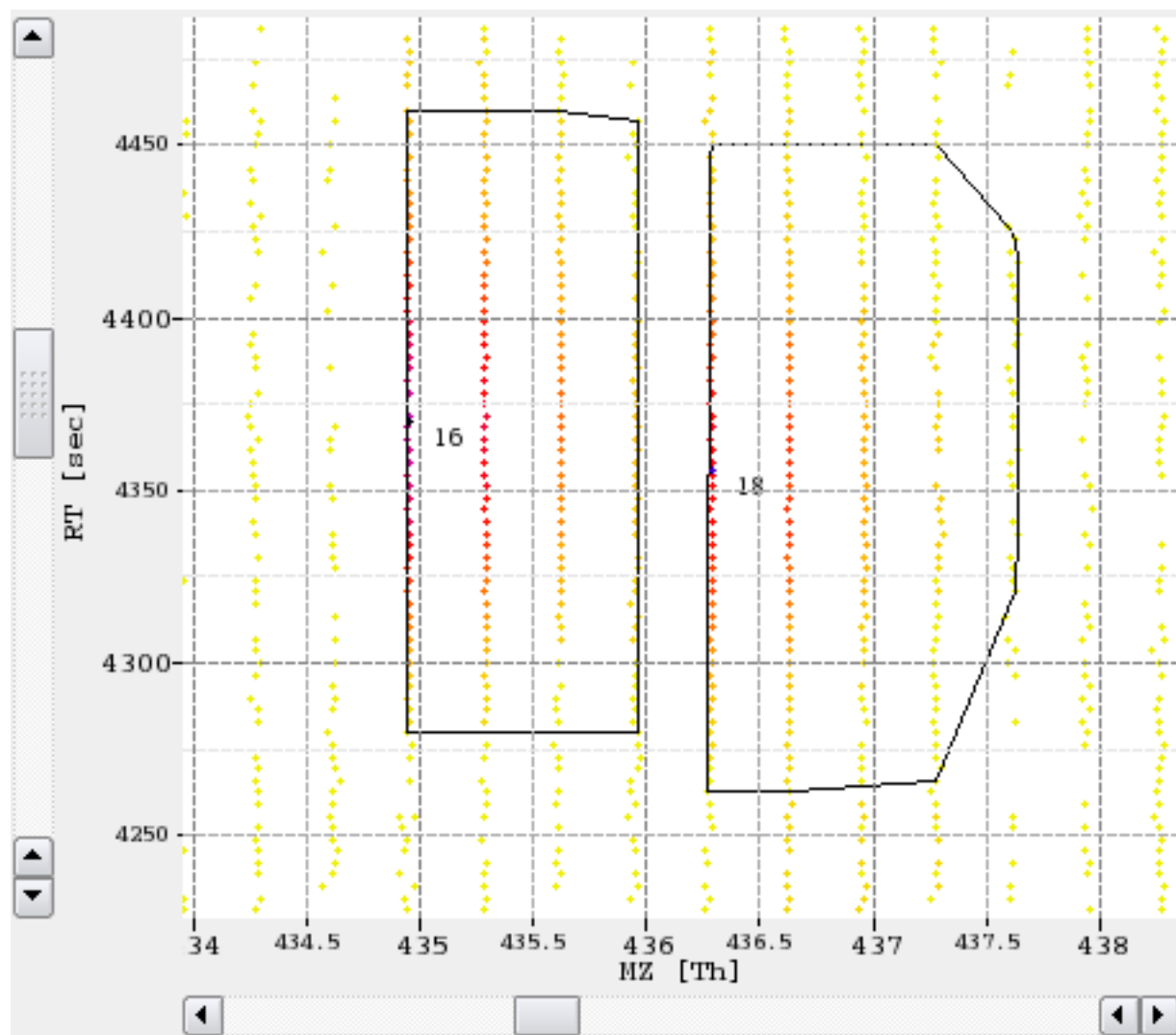
Through the **context menu** of the 1D view you can:

1. View/edit meta data.
2. Save the current layer data.
3. Change display settings.
4. Add peak annotations or arbitrary text labels.
5. Reset a performed alignment.

2D View

The 2D view is used to display peak maps and feature maps in a top-down view with color-coded intensities. Peaks and feature centroids are displayed as dots. For features, also the overall convex hull and the convex hulls of individual mass traces can be displayed. The color gradient used to encode for peak and feature intensities can be set for each layer individually.

The following example image shows a small section of a peak map and the detected features in a second layer.



In addition to the normal top-down view, the 2D view can display the projections of the data to the m/z and RT axis. This feature is mainly used to assess the quality of a feature without opening the data region in 3D view.

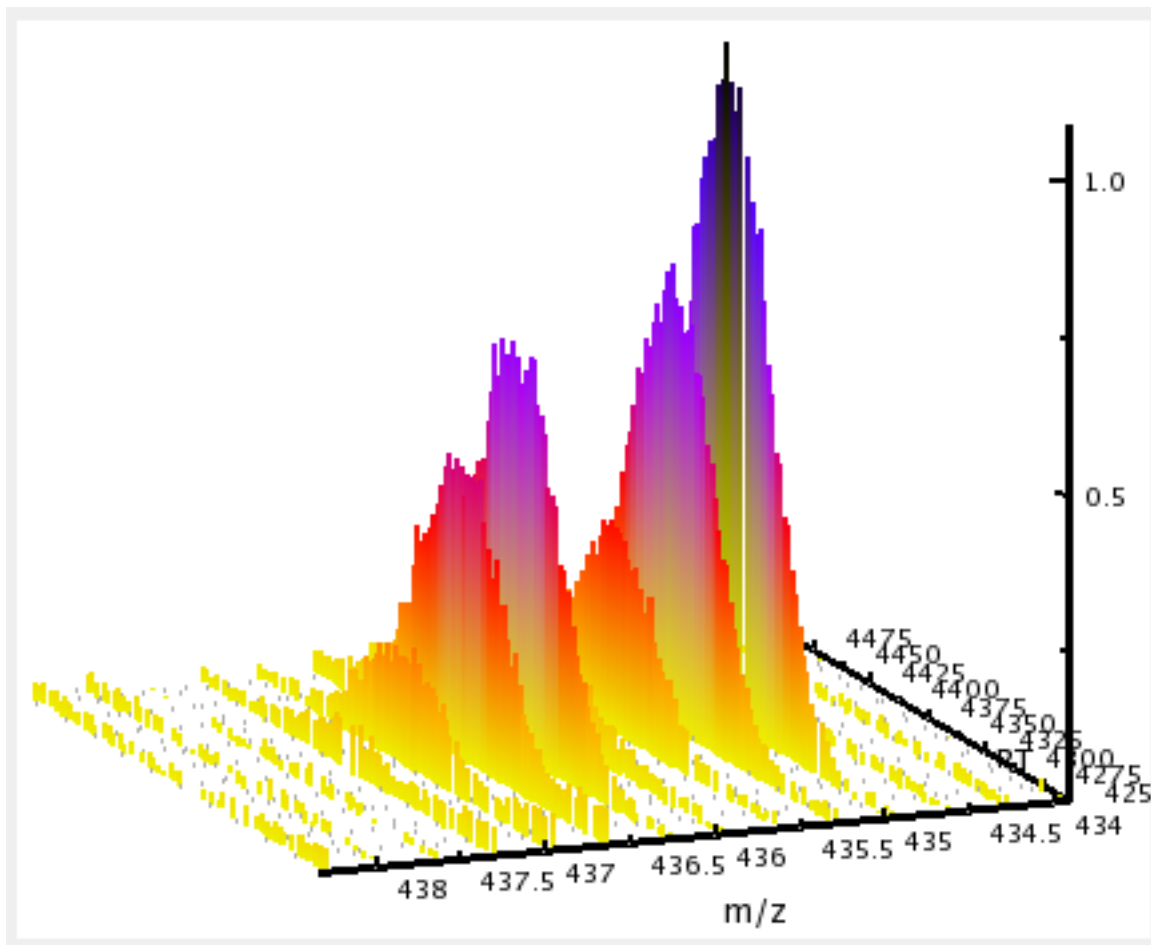
Through the **context menu** of the 2D view you can:

1. View/edit meta data
2. View survey/fragment scans in 1D view
3. View survey/fragment scans meta data
4. View the currently selected area in 3D view
5. Save the current layer data
6. Change display settings

3D View

The 3D view can only display peak maps. Its primary use is the closer inspection of a small region of the map, e.g. a single feature. In the 3D view slight intensity differences are easier to recognize than in the 2D view. The color gradient used to encode peak intensities, the width of the lines and the coloring mode of the peaks can be set for each layer individually.

The following example image shows a small region of a peak map:

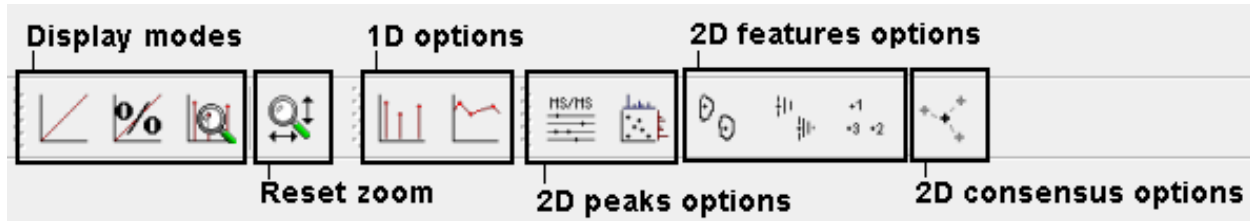


Through the **context menu**: of the 3D view you can:

1. View/edit meta data.
2. Save the current layer data.
3. Change display setting.

Display Modes and View Options in TOPPView

All of the views support several display modes and view options. Display modes determine how intensities are displayed. View options configure the view.



Display Modes

Intensity display modes determine the way peak intensities are displayed.

Linear

Normal display mode.

Percentage

In this display mode the intensities of each dataset are normalized with the maximum intensity of the dataset. This is especially useful in order to visualize several datasets that have large intensity differences. If only one dataset is opened it corresponds to the normal mode.

Snap to Maximum Intensity

In this mode the maximum currently displayed intensity is treated as if it was the maximum overall intensity.

View Options

View options configure the view of the current layer.

1D

Switching between raw data and peak mode.

2D (Peaks)

MS/MS precursor peaks can be highlighted. Projections to **m/z** and **RT** axis can be shown.

2D (Features)

Overall convex hull of the features can be displayed. Convex hulls of mass traces can be displayed (if available). A feature identifier, consisting of the feature index and an optional label can be displayed.

2D (Consensus)

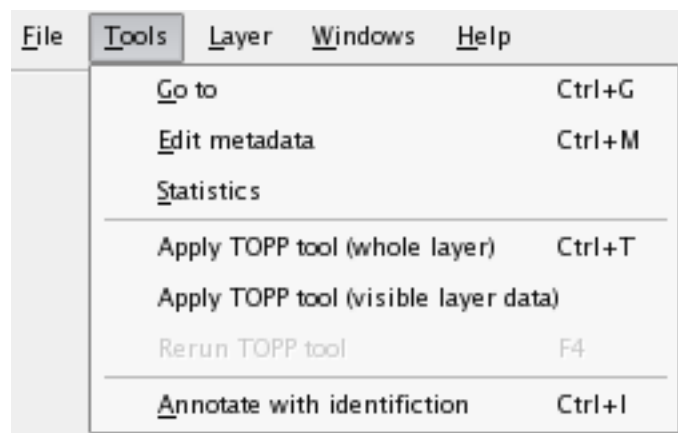
The elements of a consensus feature can be displayed.

3D

Currently there are no options for 3D view.

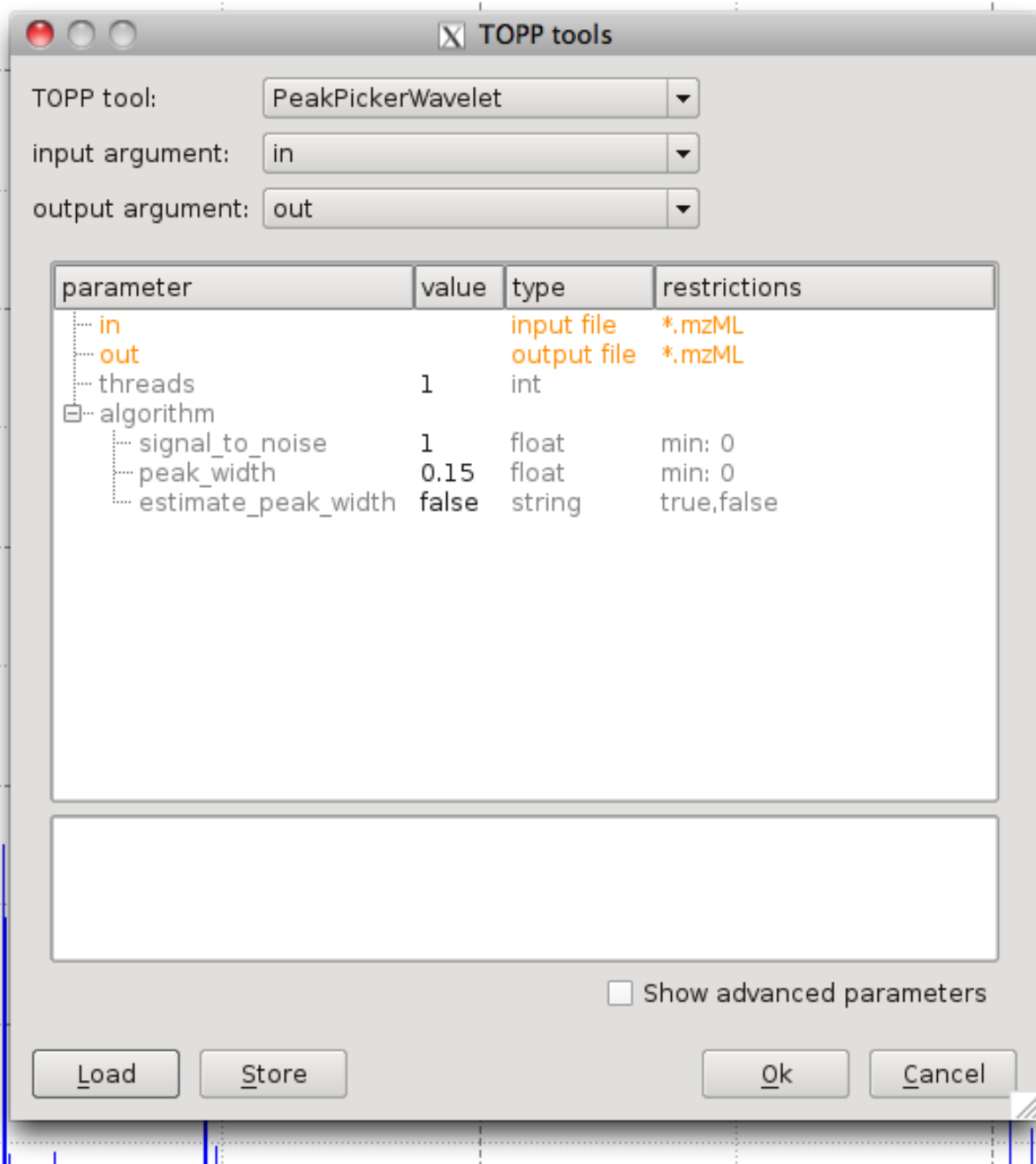
Data Analysis in TOPPView

TOPPView also offers limited data analysis capabilities for single layers, which will be illustrated in the following sections. The functionality presented here can be found in the **Tools** menu:



TOPP Tools

Single *TOPP tools* can be applied to the data of the currently selected layer or to the visible data of the current layer. The following example image shows the TOPP tools dialog:



To apply a *TOPP tool*, follow the instructions below:

1. Select a *TOPP tool* and if necessary a type.
2. Specify the command line option of the tool, that takes the input file name.
3. Specify the command line option of the tool, that takes the output file name.
4. Set the algorithm parameters manually or load them from an INI file.

Metadata

One can access the metadata, the layer is annotated with. This data comprises e.g. contact person, instrument description and sample description.

Browse in Metadata tree

- ExperimentalSettings
 - DocumentIdentifier
 - Sample MS-Sample
 - Instrument
 - IonSource**
 - MetaInfo
 - MassAnalyzer
 - MassAnalyzer
 - IonDetector
 - Software
 - MetaInfo
 - SourceFile
 - ContactPerson
 - ContactPerson
 - HPLC
 - MetaInfo

Modify ionsource information.

Order: 0

Inlet type: Direct

Ionization method: Electrospray ionisation

Polarity: Positive

Undo

OK Cancel

Tip: Identification data, e.g. from a *Mascot* run, can be annotated to the spectra or features, too. After annotation, this data is listed in the metadata.

Statistics

Statistics about peak/feature intensities and peak meta information can be displayed. For intensities, it is possible to display an additional histogram view.

Layer statistics					
	Count	Min	Max	Average	Distribution
Intensity	-	10.00	7284.19	37.27	Show
area	1041134	0.40	1210.53	7.24	
charge	1041134	0.00	0.00	0.00	
fwhm	1041134	0.10	1.92	0.19	
leftWidth	1041134	0.48	84.78	10.32	
peakShape	1041134	1.00	1.00	1.00	
rValue	1041134	0.51	1.00	0.97	
rightWidth	1041134	0.70	84.89	10.31	
signalToNoise	1041134	2.00	1147.68	10.27	

Data Editing in TOPPView

TOPPView offers editing functionality for feature layers.

After enabling the feature editing mode in the context menu of the feature layer, the following actions can be performed:

- Features can be dragged with the mouse in order to change the *m/z* and *RT* position.
- The position, intensity and charge of a feature can be edited by double-clicking a feature.
- Features can be created by double-clicking the layer background.
- Features can be removed by selecting them and pressing the DEL key.

TOPPView Hotkeys

File handling

Hotkey	Function
CTRL + O	Open file
CTRL + W	Close current window
CTRL + S	Save current layer
CTRL + SHIFT + S	Save visible data of current layer

Navigation in the data

Hotkey	Function
CTRL	Activate zoom mode
SHIFT	Activate measurement mode
Arrow keys	Translate currently shown data (1D View: Shift + Left/Right moves to the next peak in sparse data)
CTRL + +, CTRL + -	Move up and down in zoom history
Mouse wheel	Move up and down in zoom history
CTRL + G	Go to dialog
Backspace	Reset zoom
PageUp	Select previous layer
PageDown	Select next layer

Visualization options

Hotkey	Function
CTRL + R	Show/hide grid lines
CTRL + L	Show/hide axis legends
N	Intensity mode: Normal
P	Intensity mode: Percentage
S	Intensity mode: Snam-to-maximum
I	1D draw mode: peaks
R	1D draw mode: raw data
CTRL + ALT + Home	2D draw mode: increase minimum canvas coverage threshold (for raw peak scaling)
CTRL + ALT + End	2D draw mode: decrease minimum canvas coverage threshold (for raw peak scaling)
CTRL + ALT + +	2D draw mode: increase maximum point size (for raw peak scaling)
CTRL + ALT + -	2D draw mode: decrease maximum point size (for raw peak scaling)

Tip: Home on macOS keyboards is also Fn + ArrowLeft. End on macOSX keyboards is also Fn + ArrowRight.

Annotations in 1D view

Hotkey	Function
CTRL + B	Select all annotations of the current layer
DEL	Delete all currently selected annotations

Advanced

Hotkey	Function
CTRL + T	Apply TOPP tool to the current layer
CTRL + SHIFT + T	Apply TOPP tool to the visible data of the current layer
F4	Re-run TOPP tool
CTRL + M	Show layer meta information
CTRL + I	Annotate with identification results
1	Show precursor peaks (2D peak layer)
2	Show projections (2D peak layer)
5	Show overall convex hull (2D feature layer)
6	Show all convex hulls (2D feature layer)
7	Show numbers and labels (2D feature layer)
9	Show consensus elements (2D consensus layer)

Help

Hotkey	Function
F1	Show TOPPView online tutorial
SHIFT + F1	Activate <i>What's this?</i> mode

1.10 Recommended Workflow Systems

Which workflow environment to choose for running OpenMS tools?

KNIME Free, open source, desktop app. An analytics platform with workflow editor and a nice drag-and-drop user interface. Very interactive and has built-in nodes for related tasks like working with chemical structures, databases, machine learning, scripting. Distributed computing is best achieved with a KNIME server (License required) which also allows user management and a web interface to interact with workflows. In KNIME you can easily construct your own workflows or just download our ready-made creations for the most common analysis tasks.

KNIME Free, open source, desktop app. An analytics platform with workflow editor and a nice drag-and-drop user interface. Very interactive and has built-in nodes for related tasks like working with chemical structures, databases, machine learning, scripting. Distributed computing is best achieved with a KNIME server (License required) which also allows user management and a web interface to interact with workflows. In KNIME you can easily construct your own workflows or just download our ready-made creations for the most common analysis tasks.


 The logo for 'nextflow' is displayed. The word 'next' is in a green, lowercase, sans-serif font, and 'flow' is in a black, lowercase, sans-serif font. A green line or ribbon-like graphic weaves through the letters of 'next' and 'flow', connecting them.

Nextflow Script/DSL-based workflow language, executor and utilities (such as the browser based launcher and supervisor nf-tower). Automatically runs on various different cloud (AWS, Google, ...) and HPC environments (SLURM, LFS, Kubernetes, ...). It is recommended to use our ready-made nf-core compatible workflows for ease of use via the browser-based configuration and launcher.



Nextflow Script/DSL-based workflow language, executor and utilities (such as the browser based launcher and supervisor nf-tower). Automatically runs on various different cloud (AWS, Google, ...) and HPC environments (SLURM, LFS, Kubernetes, ...). It is recommended to use our ready-made nf-core compatible workflows for ease of use via the browser-based configuration and launcher.



Galaxy Server and browser-based interactive workflow editor and runner. A public server instance can be used for testing and smaller experiments. Provides nice guided tutorials.



Galaxy Server and browser-based interactive workflow editor and runner. A public server instance can be used for testing and smaller experiments. Provides nice guided tutorials.

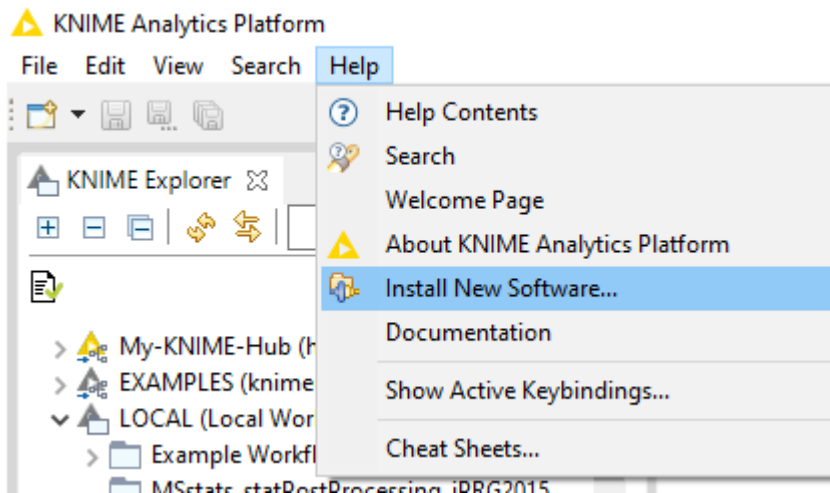
1.11 OpenMS in KNIME

KNIME is a free graphical application for creating and executing workflows. OpenMS provides a plugin, which exposes TOPP and UTILS tools to the KNIME interface.

1.11.1 Installation of KNIME and its OpenMS plugin

Installation of OpenMS in *KNIME* is platform-independent across Windows, MacOSX, and Linux.

1. Download the latest *KNIME* release from the [KNIME website](#).
2. In the full install of *KNIME* skip the following installation routine since all required plugins should be installed by default. For the standard (core) installation, follow the instructions here or in the extended *user-tutorial*.

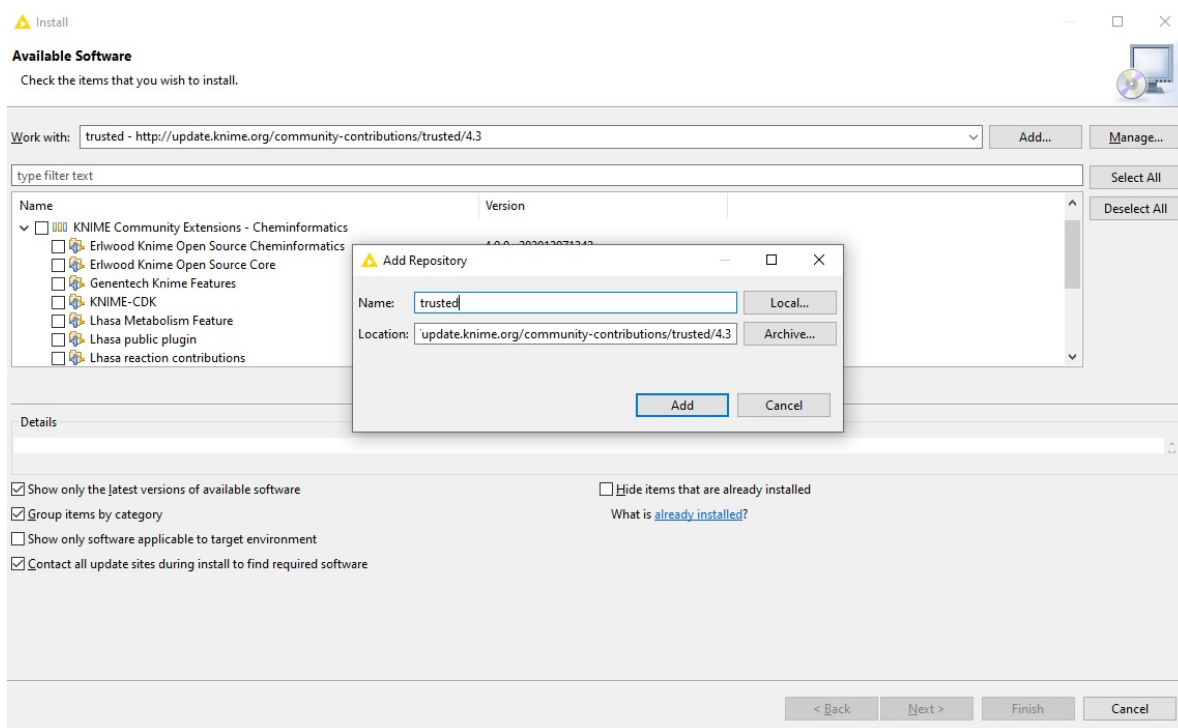


3. In KNIME click on **Help** > **Install new Software**.
4. Install the required KNIME File Handling nodes from the official KNIME Update Site (a standard entry in the update sites). Choose the update site from the **Work with:** dropdown menu.

Name: KNIME Analytics Platform 4.7.2 Update Site.

Location: <http://update.knime.org/analytics-platform/4.7.2>

5. Filter the results for **File handling** and select the *KNIME* File Handling Nodes. Click **Next** and install.



6. Now, install the actual OpenMS plugin. Next to the **Work with:** dropdown menu, click on **Add...** In the opening dialog fill in at least one of the following additional Update Sites (if not already present):

- **Recommended:**

Name: KNIME Community Contributions (Stable)

Location: <http://update.knime.org/community-contributions/trusted/4.7.2>

- **Unstable:**

Name: KNIME Nightly Community Contributions (trunk)

Location: <http://update.knime.org/community-contributions/trunk>

- **OpenMS-internal, unstable:**

Name: OpenMS nightly update site

Location: <https://abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/nightly/>

7. Use the search or navigate to **KNIME Community Contributions – Bioinformatics & NGS** and select **OpenMS**. Then click **Next** and follow the installation instructions. A restart of KNIME might be necessary afterward. On Windows, if prompted to install additional requirements like the Microsoft Visual Studio Redistributable for the conversion software ProteoWizard that is packaged with our plugin.

8. After a restart of KNIME the OpenMS nodes will be available in your Node Repository (panel on the lower left) under **Community Nodes**.

Creating workflows with KNIME

Download Introduction to OpenMS in KNIME [user tutorial](#) containing hands-on training material covering also basic usage of KNIME. See the official [KNIME Getting Started Guide](#) for a more in-depth view of the KNIME functionality besides OpenMS.

If you face any issues, please [contact us](#) and specifically for the usage of OpenMS in KNIME, the KNIME community contribution [forum](#).

Creating your own Generic KNIME Nodes

To create your own generic KNIME nodes, refer to their [documentation](#).

1.11.2 Ready-made OpenMS KNIME workflows

You can get ready-made KNIME workflows and workflow components with OpenMS nodes from our [community hub](#) which is also embedded below. You can easily drag-and-drop workflows into your opened KNIME analytics platform. For more, e.g., thirdparty OpenMS workflows use the search bar on the hub and [search for “openms”](#).

1.11.3 OpenMS KNIME User Tutorial

General Remarks

- This handout will guide you through an introductory tutorial for the OpenMS/TOPP software package¹.
- OpenMS^{2,3} 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 Pipeline”)⁴ can be understood as small building blocks of arbitrarily complex data analysis workflows.
- In order to facilitate workflow construction, OpenMS was integrated into KNIME⁵, the Konstanz Information Miner, an open-source integration platform providing a powerful and flexible workflow system combined with advanced data analytics, visualization, and report capabilities. Raw MS data as well as the results of data processing using TOPP can be visualized using TOPPView⁶.
- This tutorial was designed for use in a hands-on tutorial session but can also be worked through at home using the online resources. You will become familiar with some of the basic functionalities of OpenMS/TOPP, TOPPView, as well as KNIME and learn how to use a selection of TOPP tools used in the tutorial workflows.
- If you are attending the tutorial and received a USB stick, all sample data referenced in this tutorial can be found in the C:Example_Data folder, on the USB stick, or released online on our [Archive](#).

¹ OpenMS, OpenMS home page [online].

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 opensource software framework for mass spectrometry., BMC bioinformatics 9(1) (2008), doi:10.1186/1471-2105-9-163. 7, 83

³ H. L. Röst, T. Sachsenberg, S. Aiche, C. Bielow, H. Weisser, F. Aicheler, S. Andreotti, H.-C. Ehrlich, P. Gutenbrunner, E. Kenar, et al., OpenMS: a flexible open-source software platform for mass spectrometry data analysis, Nature Methods 13(9), 741–748 (2016). 7

⁴ 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). 7, 83

⁵ M. R. Berthold, N. Cebon, 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 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. 7

Getting Started

Installation

Before we get started, we will install OpenMS with its viewer TOPPView, KNIME and the OpenMS KNIME plugin. If you take part in a live training session you will have likely received an USB stick from us that contains the required data and software. If we provide laptops with the software you may of course skip the installation process and continue reading the next section. If you are doing this tutorial online, choose online in the following tab(s).

Online

If you are working through this tutorial at home/online, proceed with the following steps:

- Download and install OpenMS using the installation instructions for the [OpenMS tools](#).

Note: To install the graphical application, please use the downloadable installer for your platform, not conda, nor docker.

- Download and install [KNIME](#)

USB Stick

Please choose the directory that matches your operating system and execute the installer.

For Windows, you run:

- The OpenMS installer: WindowsOpenMS-3.0.0-Win64.exe
- The KNIME installer: WindowsKNIME-4.7.2-Installer-64bit.exe

On macOS, you run:

- The OpenMS installer: MacOpenMS-3.0.0-macOS.dmg
- The KNIME installer: Macknime_3.0.0.app.macosx.cocoa.x86_64.dmg

On Linux, you can extract KNIME to a folder of your choice and for TOPPView you need to install OpenMS via your package manager or build it on your own with our [build instructions](#).

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 MSConvert, which is part of a ProteoWizard installation. All files used in this tutorial have already been converted to mzML by us, so you do not need to perform the data conversion yourself. However, we provide a small raw file so you can try the important step of raw data conversion for yourself.

Note: The OpenMS installation package for Windows automatically installs ProteoWizard, so you do not need to download and install it separately. Due to restrictions from the instrument vendors, file format conversion for most formats is only possible on Windows systems. In practice, performing the conversion to mzML on the acquisition PC connected to the instrument is usually the most convenient option.

To convert raw data to mzML using ProteoWizard you can either use MSConvertGUI (a graphical user interface) or `msconvert` (a simple command line tool).

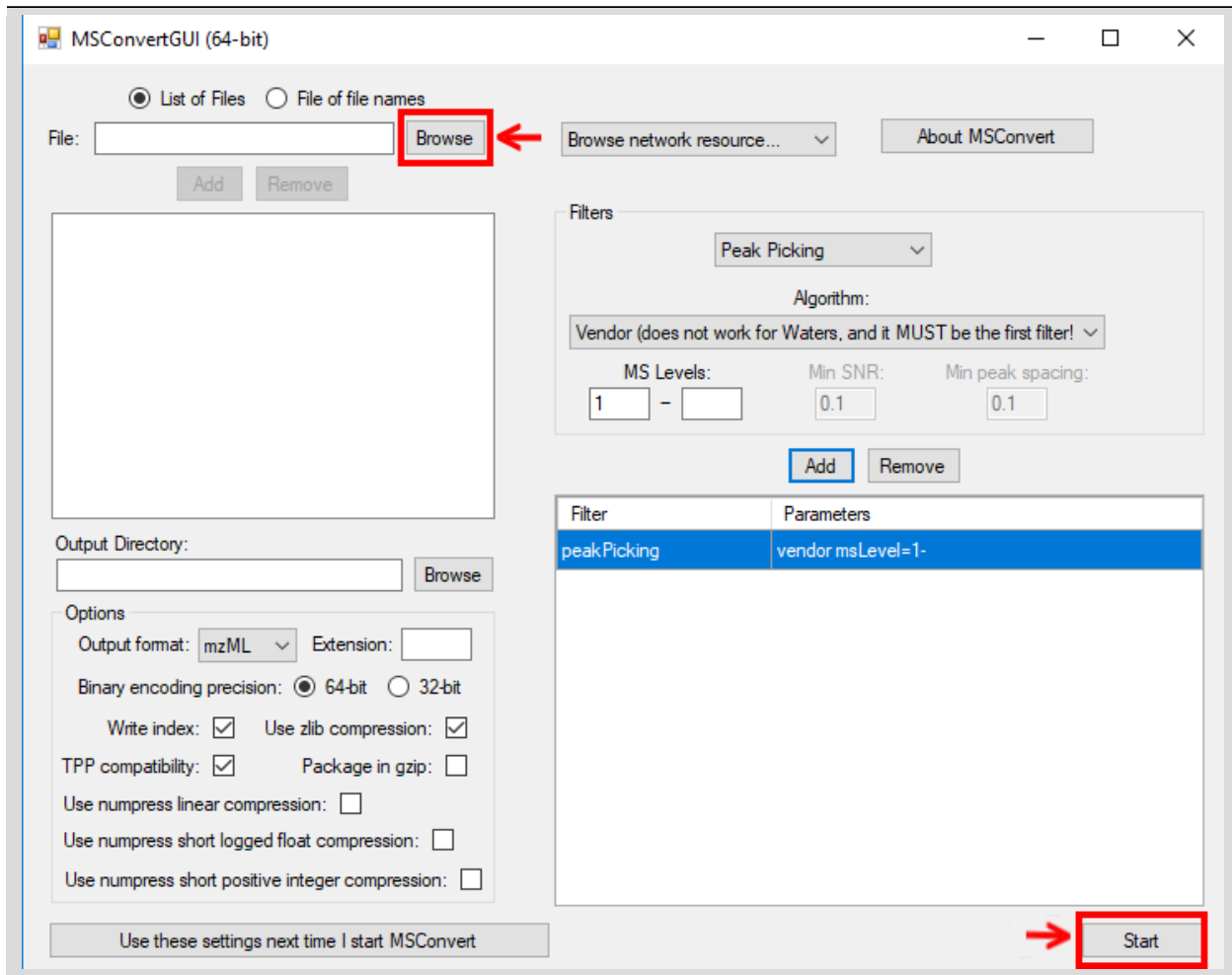


Figure 1: MSConvertGUI (part of ProteoWizard), allows converting raw files to mzML. Select the raw files you want to convert by clicking on the browse button and then on Add. Default parameters can usually be kept as-is. To reduce the initial data size, make sure that the `peakPicking` filter (converts profile data to centroided data (see Fig. 2)) is listed, enabled (true) and applied to all MS levels (parameter "1-"). Start the conversion process by clicking on the Start button.

Both tools are available in: `C:\Program Files\OpenMS-3.0.0\share\OpenMS\THIRDPARTY\pwiz-bin`.

You can find a small RAW file on the USB stick `C:\Example_Data\Introduction\datasets\raw`.

MSConvertGUI

MSConvertGUI (see Fig. 1) exposes the main parameters for data conversion in a convenient graphical user interface.

msconvert

The `msconvert` command line tool has no graphical user interface but offers more options than the application MSConvertGUI. Additionally, since it can be used within a batch script, it allows converting large numbers of files and can be much more easily automatized. To convert and pick the file `raw_data_file.RAW` you may write:

```
msconvert raw_data_file.RAW --filter "peakPicking true 1-"
```

in your command line.

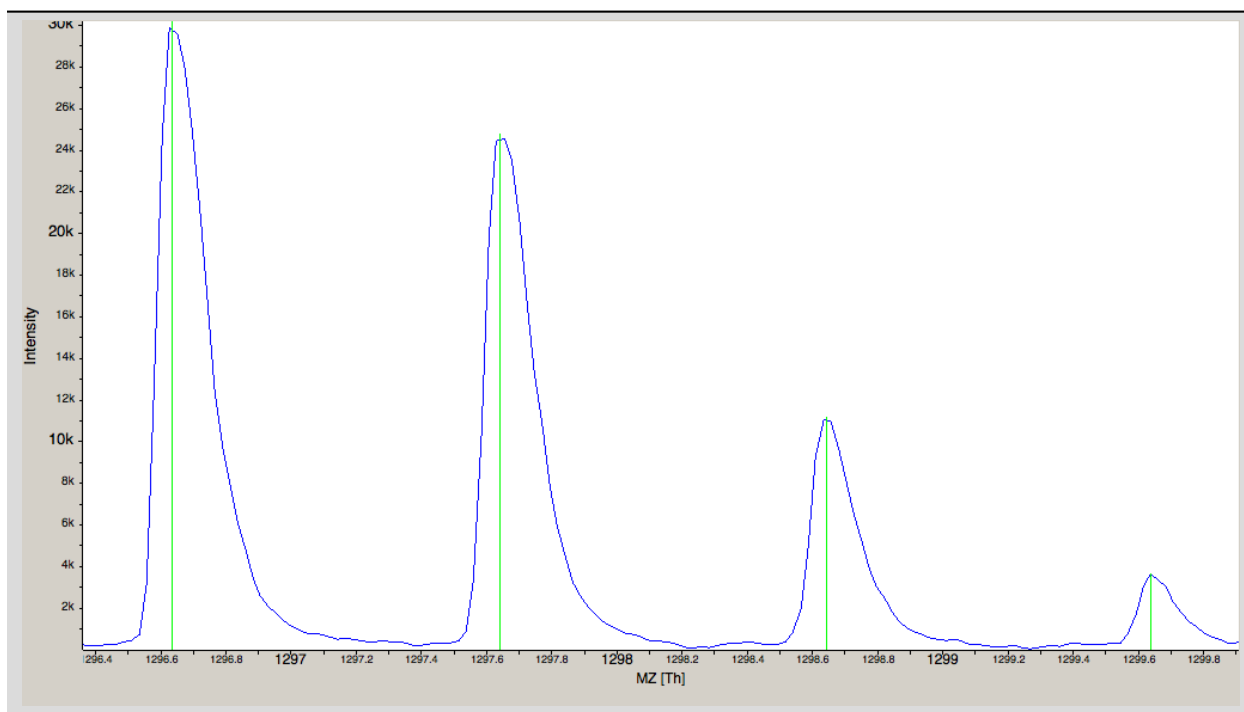


Figure 2: The amount of data in a spectra is reduced by peak picking. Here a profile spectrum (blue) is converted to centroided data (green). Most algorithms from this point on will work with centroided data.

To convert all RAW files in a folder may write:

```
msconvert *.RAW -o my_output_dir
```

Note: To display all options you may type `msconvert --help`. Additional information is available on the ProteoWizard web page.

ThermoRawFileParser

Recently the open-source platform independent ThermoRawFileParser tool has been developed. While Proteowizard and MSConvert are only available for Windows systems this new tool allows to also convert raw data on Mac or Linux.

Note: To learn more about the ThermoRawFileParser and how to use it in KNIME see [A minimal workflow](#).

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. Note that conceptually, there are no differences in visualizing metabolomic or proteomic data. Here, we inspect a simple proteomic measurement:

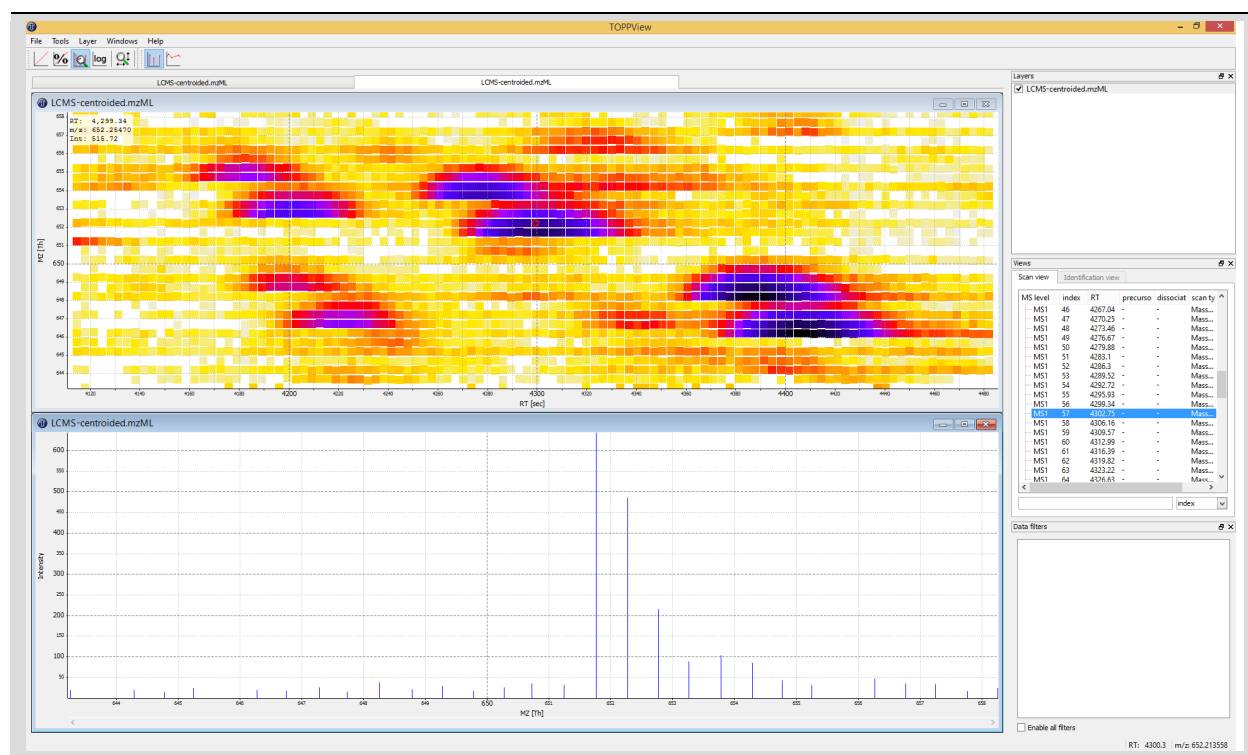


Figure 3: TOPPView, the graphical application for viewing mass spectra and analysis results. Top window shows a small region of a peak map. In this 2D representation of the measured spectra, signals of eluting peptides are colored according to the raw peak intensities. The lower window displays an extracted spectrum (=scan) from the peak map. On the right side, the list of spectra can be browsed.

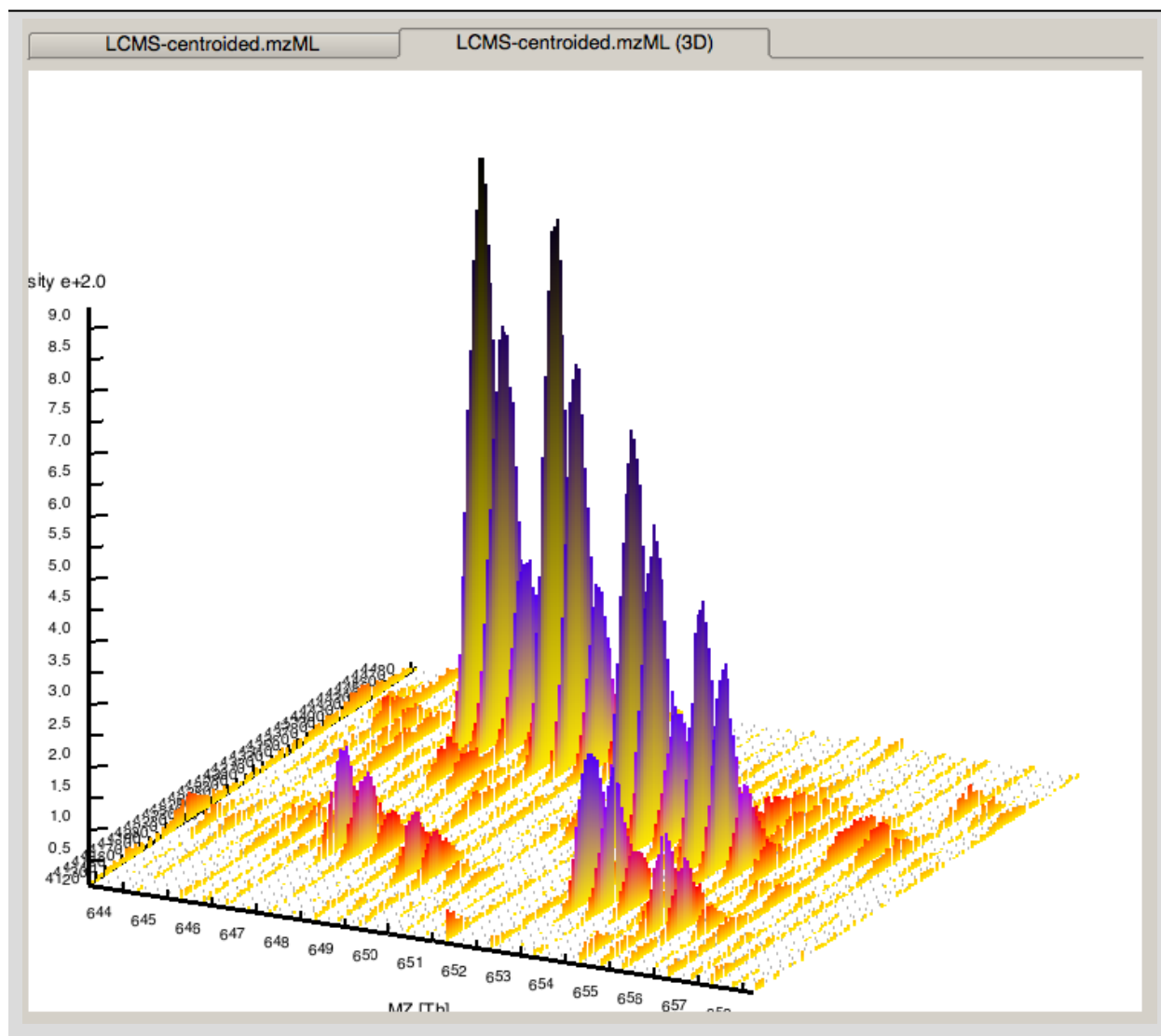


Figure 4: 3D representation of the measured spectra, signals of eluting peptides are colored according to the raw peak intensities.

- Start TOPPView (see Windows' Start-Menu or ApplicationsOpenMS-3.0.0 on macOS)
- Go to **File > Open File**, navigate to the directory where you copied the contents of the USB stick to, and select Example_DataIntroductiondatasetssmallvelos005614.mzML. This file contains only a reduced LC-MS map of a label-free proteomic platelet measurement recorded on an Orbitrap velos. The other two mzML files contain technical replicates of this experiment. First, we want to obtain a global view on the whole LC-MS map - the default option Map view 2D is the correct one and we can click the Ok button.
- Play around.
- Three basic modes allow you to interact with the displayed data: scrolling, zooming and measuring:
 - **Scroll mode**
 - * Is activated by default (though each loaded spectra file is displayed zoomed out first, so you do not need to scroll).
 - * Allows you to browse your data by moving around in RT and m/z range.

- * When zoomed in, you can scroll through the spectra. Click-drag on the current view.
- * Arrow keys can be used to scroll the view as well.

– Zoom mode

- * 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 zoom in and out.
- * 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).
- * Pressing backspace ← zooms out to show the full LC-MS map (and also resets the zoom history).

– Measure mode

- * It is activated using the Shift 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 mode** and examine your data in 3D mode (see Fig. 4).
- Go back to the 2D view. In 2D mode, visualize your data in different intensity normalization modes, use linear, percentage, snap and log-view (icons on the upper left tool bar). You can hover over the icons for additional information.

Note: On macOS, 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.

Dependent on your data MS/MS spectra can be visualized as well (see Fig.5) . You can do so, by double-click on the MS/MS spectrum shown in scan view

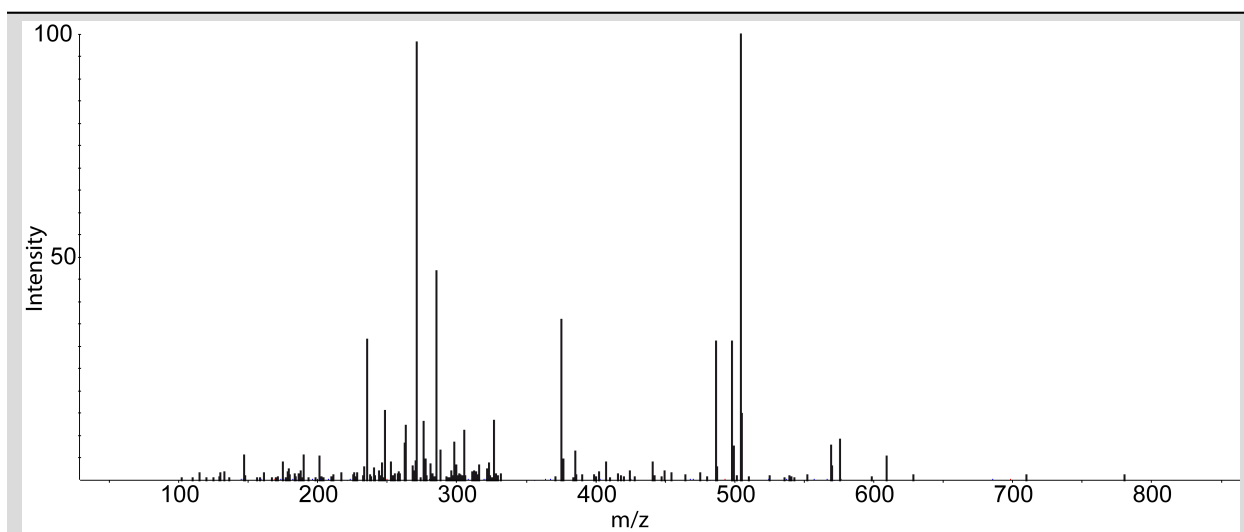


Figure 5: MS/MS spectrum

Introduction to KNIME/OpenMS

Using OpenMS in combination with KNIME, you can create, edit, open, save, and run workflows that combine 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, to make it more difficult to create an invalid workflow. Throughout most parts of this tutorial, you will use KNIME to create and execute workflows. The first step is to become familiar with KNIME. Additional information on the basic usage of KNIME can be found on the KNIME [Getting Started page](#). However, the most important concepts will also be reviewed in this tutorial.

Plugin and dependency

Before we can start with the tutorial, we need to install all the required extensions for KNIME. Since KNIME 3.2.1, the program automatically detects missing plugins when you open a workflow but to make sure that the right source for the OpenMS plugin is chosen, please follow the instructions here.

Required KNIME plugins

First, we install some additional extensions that are required by our OpenMS nodes or used in the Tutorials for downstream processing, visualization or reporting.

1. In KNIME, click on **Help > Install New Software**.
2. From the ‘**Work with:**’ drop-down list, select the *update site* ‘KNIME 4.6 - <https://update.knime.com/community-contributions/trusted/4.6>’
3. Now select the following KNIME core plugins from the KNIME & Extensions category
 - KNIME Base Chemistry Types & Nodes
 - KNIME Chemistry Add-Ons
 - KNIME Interactive R Statistics Integration
 - KNIME Report Designer
 - KNIME SVG Support
4. Click on **Next** and follow the instructions (you may but don’t need to restart KNIME now).
5. Click again on **Help > Install New Software**
6. From the ‘**Work with:**’ drop-down list, select the *update site* ‘KNIME Community Extensions (Trusted) - <https://update.knime.com/community-contributions/trusted/4.6>’
7. Now select the following plugin from the “KNIME Community Contributions - Cheminformatics” category
 - RDKit KNIME integration
8. Click on **Next** and follow the instructions and after a restart of KNIME the dependencies will be installed.

R programming language and its KNIME integration

In addition, we need to install R for the statistical downstream analysis. Choose the directory that matches your operating system, double-click the R installer and follow the instructions. We recommend to use the default settings whenever possible. On macOS you also need to install XQuartz from the same directory.

Afterwards open your R installation. If you use Windows, you will find an "R x64 3.6.X" icon on your desktop. If you use macOS, you will find R in your Applications folder. In R, type the following lines (you might also copy them from the file Rinstall_R_packages.R on the USB stick):

```
install.packages('Rserve',, "http://rforge.net/", type="source")
install.packages("Cairo")

install.packages("devtools")
install.packages("ggplot2")
install.packages("ggfortify")

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install()
BiocManager::install(c("MSstats"))
```

In KNIME, click on **KNIME > Preferences**, select the category **KNIME > R** and set the "Path to R Home" to your installation path. You can use the following settings, if you installed R as described above:

- Windows: C:\Program Files\R\R-3.6.X' (where X is the version you used to install the above libraries)
- macOS: /Library/Frameworks/R.framework/Versions/3.6/Resources

KNIME OpenMS plugin

You are now ready to install the OpenMS nodes.

- In KNIME, click on **Help > Install New Software**

You now have to choose an *update site* to install the OpenMS plugin from. Which *update site* to choose depends on if you received an USB stick in a hands-on Tutorial or if you are doing this Tutorial online.

Online

To install the OpenMS KNIME plugin from the internet, do the following:

1. From the 'Work with:' drop-down list, select the *update site* 'KNIME Community Extensions (Trusted)' - <https://update.knime.com/community-contributions/trusted/4.6>
2. Now select the following plugin from the "KNIME Community Contributions - Bioinformatics & NGS" category
 - OpenMS
3. Click on **Next** and follow the instructions and after a restart of KNIME the OpenMS nodes will be available in the Node repository under "Community Nodes".

Note: If this does not work for you, report it and while waiting for a reply/fix, try to use an *update site* of an older KNIME version by editing the KNIME version number in the URL or by using our unofficial *update site* at <https://update.knime.com/community-contributions/trusted/4.6>

[//abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/release/latest](http://abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/release/latest)

USB

We included a custom KNIME update site to install the OpenMS KNIME plugins from the USB stick. If you do not have a stick available, please see below.

- 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: OpenMS 3.0.0 UpdateSite

Location: file:/KNIMEUpdateSite/3.0.0/

- After pressing **OK** KNIME will show you all the contents of the added Update Site.

Note: From now on, you can use this repository for plugins in the **Work with:** drop-down list.

- Select the OpenMS nodes in the "Uncategorized" category and click **Next**.
- Follow the instructions and after a restart of KNIME the OpenMS nodes will be available in the Node repository under "Community Nodes".

Online experimental

To install the nightly/experimental version of the OpenMS KNIME plugin from the internet, do the following:

- 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: OpenMS 3.0.0 UpdateSite

Location: <https://abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/nightly/>

- After pressing **OK** KNIME will show you all the contents of the added Update Site.

Note: From now on, you can use this repository for plugins in the **Work with:** drop-down list.

- Select the OpenMS nodes in the "Uncategorized" category and click **Next**.
- Follow the instructions and after a restart of KNIME the OpenMS nodes will be available in the Node repository under "Community Nodes".

KNIME concepts

A workflow is a sequence of computational steps applied to a single or multiple input data to process and analyze the data. In KNIME such workflows are implemented graphically by connecting so-called nodes. A node represents a single analysis step in a workflow. Nodes have input and output ports where the data enters 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 of data in KNIME are tables (similar to an excel sheet). Ports that accept tables are marked with a small triangle. For OpenMS nodes, we use a different port type, so called file ports, representing complete files. Those ports are marked by a small blue box. Filled blue boxes represent mandatory inputs and empty blue boxes optional inputs. The same holds for output ports, despite you can deactivate them in the

configuration dialog (double-click on node) under the **OutputTypes** tab. After execution, deactivated ports will be marked with a red cross and downstream nodes will be inactive (not configurable).

A typical OpenMS workflow in KNIME can be divided in two conceptually different parts:

- Nodes for signal and data processing, filtering and data reduction. Here, files are passed between nodes. Execution times of the individual steps are typically longer for these types of nodes as they perform the main computations.
- Downstream statistical analysis and visualization. Here, tables are passed between nodes and mostly internal KNIME nodes or nodes from third-party statistics plugins are used. The transfer from files (produced by OpenMS) and tables usually happens with our provided Exporter and Reader nodes (e.g. MzTabExporter followed by MzTabReader).

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 fails, the node will switch to the red state. Other anomalies and warnings like missing information or empty results will be presented with a yellow exclamation mark above the traffic light. Most nodes will be configured as soon as all input ports are connected. Some nodes need to know about the output of the predecessor and may stay red until the predecessor was executed. If nodes still remain in a red state, probably additional parameters have to be provided in the configuration dialog that can neither be guessed from the data nor filled with sensible defaults. In this case, or if you want to customize the default configuration in general, you can open the configuration dialog of a node with a double-click on the node. For all OpenMS nodes you will see a configuration dialog like the one shown in below figure.

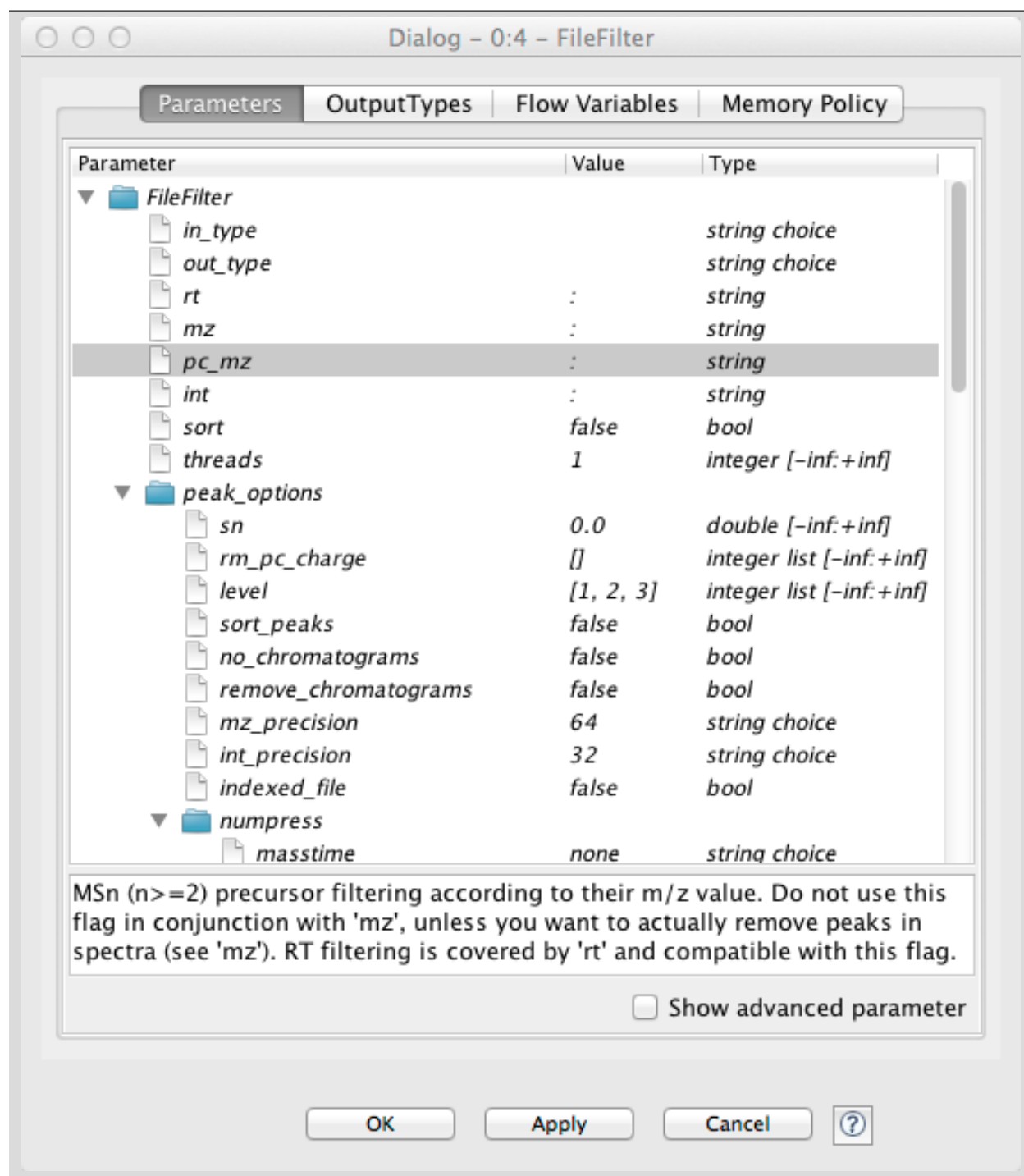


Figure 6: Node configuration dialog of an OpenMS node

Tip: OpenMS distinguishes between normal parameters and advanced parameters. Advanced parameters are by default hidden from the users since they should only rarely be customized. In case you want to have a look at the parameters or need to customize 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. Afterwards the parameters are shown in a light gray color.

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. Please also note the tabs on the top of the configuration dialog. In the case of OpenMS nodes, there will be another tab called OutputTypes. It contains dropdown menus for every output port that let you select the output filetype that you want the node to return (if the tool supports it). For optional output ports you can select Inactive such that the port is crossed out after execution and the associated generation of the file and possible additional computations are not performed. Note that this will deactivate potential downstream nodes connected to this port.

Overview of the graphical user interface

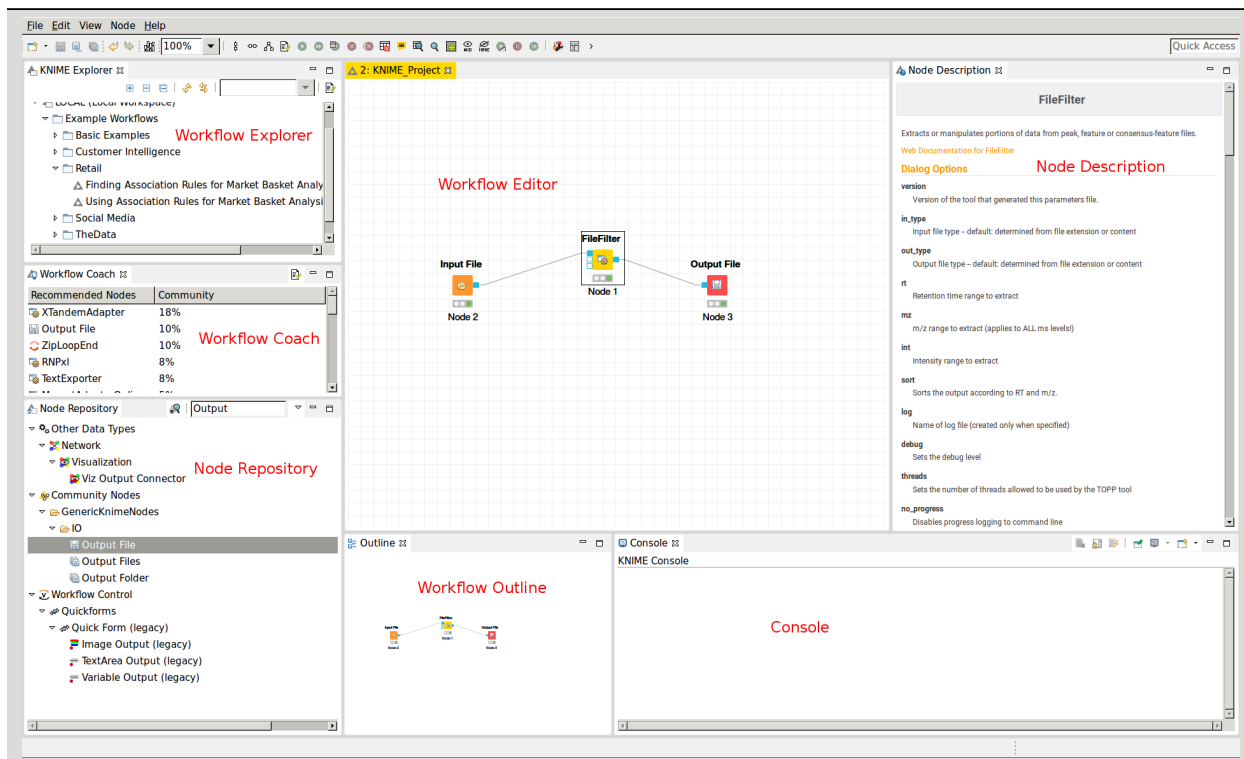


Figure 7: The KNIME workbench

The graphical user interface (GUI) of KNIME consists of different components or so-called panels that are shown in above image. We will briefly introduce the individual panels and their purposes below.

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". For quick creation of a workflow, note that double-clicking on a node in the repository automatically connects it to the selected node in the workbench (connecting all the inputs with as many fitting outputs of the last node). Manually, nodes can be connected by clicking on the output port of one node and dragging the edge until releasing the mouse at the desired input port of the next node. Deletions are possible by selecting nodes and/or edges and pressing DEL or Fn + Backspace depending on your OS and settings. Multiselection happens via dragging rectangles with the mouse or adding elements to the selection by clicking them while holding down Ctrl.

KNIME 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 choosing **New KNIME Workflow** from the appearing context menu. Remember to save your workflow often with the Ctrl + S shortcut.

Workflow Coach (since KNIME 3.2.1)

Shows a list of suggested following nodes, based on the last added/clicked nodes. When you are not sure which node to choose next, you have a reasonable suggestion based on other users behavior there. Connect them to the last node with a double-click.

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 Node > OpenMS** Nodes for managing files (e.g., Input Files or Output Folders) can be found in **Community Nodes > GenericKnimeNode**. 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. You can adjust the zoom level of the explorer by adjusting the percentage in the toolbar at the top of KNIME.

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 a warning 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 and especially their in- and outputs, such that you know what types of data nodes may produce or expect. For OpenMS nodes you will also find a link to the tool page of the online documentation.

Creating workflows

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

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 *knwf* file containing all the information on nodes, their connections, and their parameter configuration.

Those *knwf* 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 on the USB Stick. Additionally, the workflow files can be found on workflow downloads. 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 *knwf* file and after that double-click it in your KNIME Workflow repository to open it.

Duplicating workflows

In this 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, save it, close it and follow the next 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, too.

A minimal workflow

Let us now start with the creation of a 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. This minimal workflow can also be used to check if all requirements are met and that your system is compatible.

- 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 Node > 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.

Tip: 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 below image. FileInfo can produce two different kinds of output files.

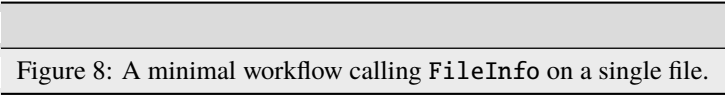


Figure 8: A minimal workflow calling `FileInfo` on a single file.

- 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 `Example_DataIntroductiondatasetsstinyvelos005614.mzML` and click **Open**. Afterwards close the dialog by clicking **Ok**.

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 $\uparrow + F7$ (shift key + F7; or the button with multiple green triangles in the KNIME Toolbar) 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.

Workflows are typically constructed to process a large number of files automatically. As a simple example, consider you would like to convert multiple Thermo Raw files into the mzML format. 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.
- 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 **Example_Data > Introduction > datasets > 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 **FileConverter** node, the first output port of the **FileConverter** node to the first input port of the **ZipLoopEnd** node, and the first output port of the **ZipLoopEnd** node to the **Output Folder** node (NOT to the Output File).

The complete workflow is shown in below figure.

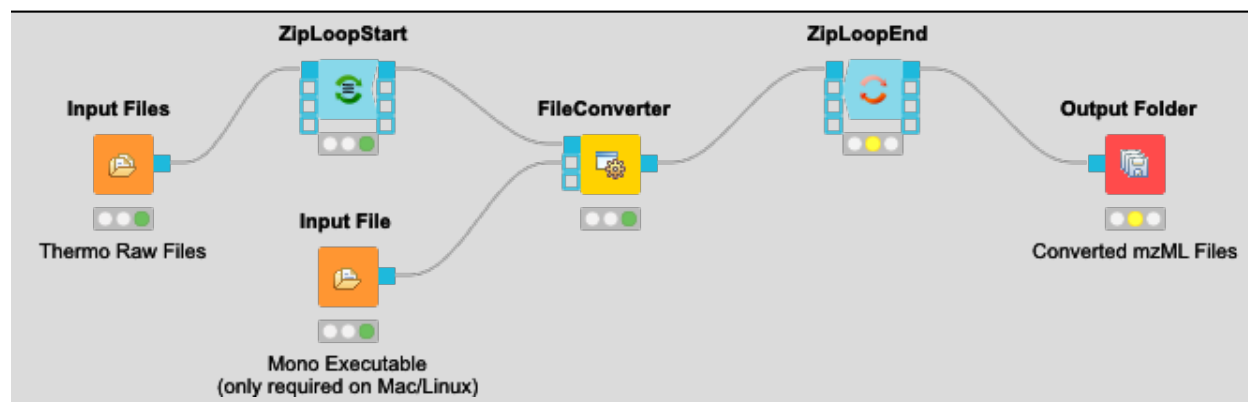


Figure 9: A minimal workflow calling the FileConverter on multiple Thermo Raw files in a loop

Execute the workflow and inspect the output as before.

In case you had trouble to understand what **ZipLoopStart** and **ZipLoopEnd** do, here is a brief explanation:

- The **Input Files** node passes a list of files to the ZipLoopStart node.
- The ZipLoopStart node takes the files as input, but passes the single files sequentially (that is: one after the other) to the next node.
- The ZipLoopEnd collects the single files that arrive at its input port. After all files have been processed, the collected files are passed again as file list to the next node that follows.

Digression: Working with chemical structures

Metabolomics analyses often involve working with chemical structures. Popular cheminformatic toolkits such as RDKit⁷ or CDK⁸ are available as KNIME plugins and allow us to work with chemical structures directly from within KNIME. In particular, we will use KNIME and RDKit to visualize a list of compounds and filter them by predefined sub-structures. Chemical structures are often represented as SMILES (Simplified molecular input line entry specification), a simple and compact way to describe complex chemical structures as text. For example, the chemical structure of L-alanine can be written as the SMILES string C[C@H](N)C(O)=O. As we will discuss later, all OpenMS tools that perform metabolite identification will report SMILES as part of their result, which can then be further processed and visualized using RDKit and KNIME.

⁷ RDKit: Open-source cheminformatics, <http://www.rdkit.org>, [Online; accessed 31-August-2018]. 25

⁸ C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttmann, and E. Willighagen, The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics, Journal of Chemical Information and Computer Sciences 43(2), 493–500 (2003), PMID: 12653513, doi:10.1021/ci025584y. 25

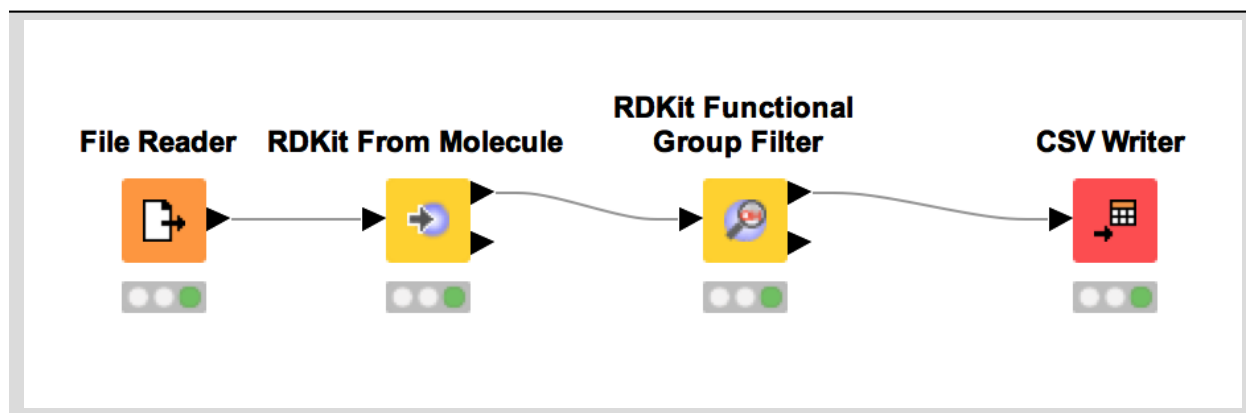


Figure 10: Workflow to visualize a list of SMILES strings and filter them by predefined substructures

Perform the following steps to build the workflow shown in the above figure. You will use this workflow to visualize a list of SMILES strings and filter them by predefined substructures:

- Add the node **File Reader**, open the node configuration dialog and select the file `smiles.csv`. This file has been exported from the Human Metabolome Database (HMDB) and contains the portion of the human metabolome that has been detected and quantified. The file preview on the bottom of the dialog shows that each compound is given by its HMDB accession, compound name, and SMILES string. Click on the column header **SMILES** to change its properties. Change the column type from **string** to **smiles** and close the dialog with **Ok**. Afterwards the **SMILES** column will be visualized as chemical structures instead of text directly within all **KNIME** tables.
- Add the node **RDKit From Molecule** and connect it to the **File Reader**. This node will use the provided SMILES strings to add an additional column that is required by RDKit.
- Add the node **RDKit Functional Group Filter** and open the node configuration dialog. You can use this dialog to filter the compounds by any combination of functional groups. In this case we want to find all compounds that contain at least one aromatic carboxylic acid group. To do this, set this group as active and choose `'C(=O)O'` and `'1'`.
- Connect the first output port (Molecules passing the filter) to a **CSV Writer** node to save the filtered metabolites to a file. Right click **RDKit Functional Group Filter** and select the view 'Molecules passing the filter' to inspect the selected compounds in KNIME. How many compounds pass the chosen filter, see below figure.

The following figure shows resulting list of compounds that contains at least one aromatic carboxylic acid group.

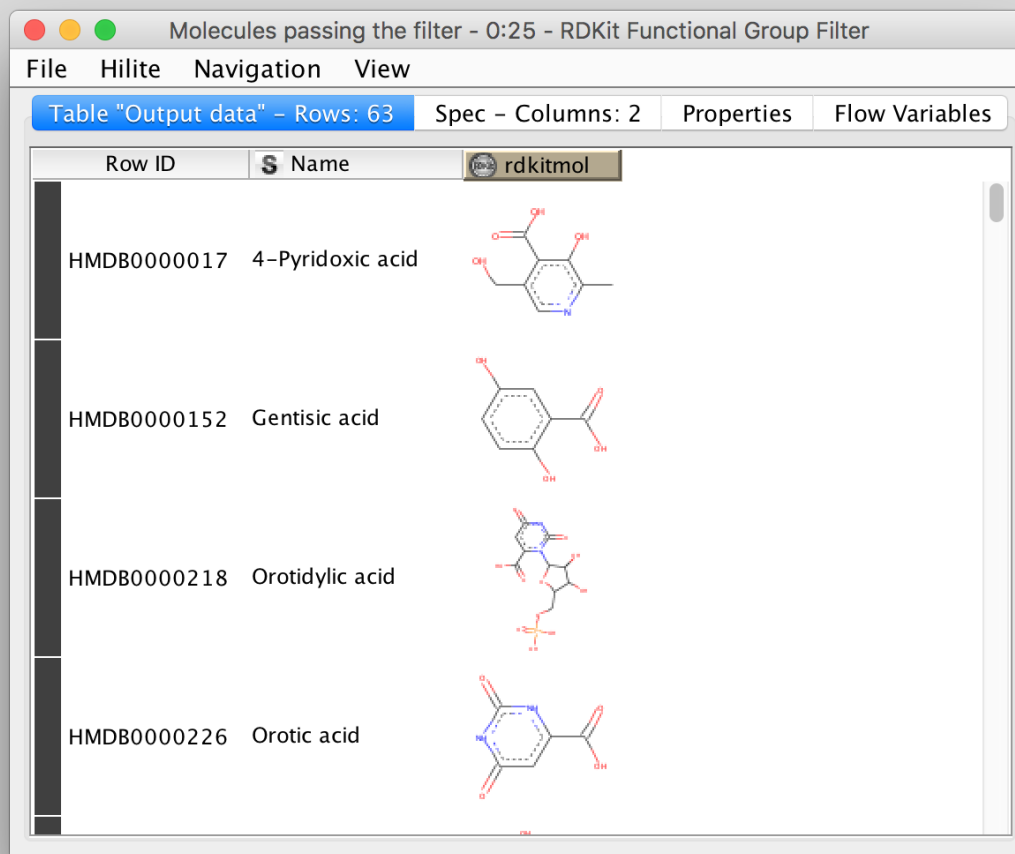


Figure 11: Resulting list of compounds that contains at least one aromatic carboxylic acid group.

Advanced topic: Metanodes

Workflows can get rather complex and may contain dozens or even hundreds of nodes. KNIME provides a simple way to improve handling and clarity of large workflows:

Metanodes allow to bundle several nodes into a single **Metanode**.

Task

Select multiple nodes (e.g. all nodes of the ZipLoop including the start and end node). To select a set of nodes, draw a rectangle around them with the left mouse button or hold Ctrl to add/remove single nodes from the selection.

Tip

There is a **Select Loop** option when you right-click a node in a loop, that does exactly that for you. Then, open the context menu (right-click on a node in the selection) and select **Create Metanode**. Enter a caption for the **Metanode**.

The previously selected nodes are now contained in the **Metanode**. Double-clicking on the **Metanode** will display the contained nodes in a new tab window.

Task

Create the Metanode to let it behave like an encapsulated single node. First select the **Metanode**, open the context menu (right-click) and select **Metanode > Wrap**. The differences between Metanodes and their wrapped counterparts are marginal (and only apply when exposing user inputs and workflow variables). Therefore we suggest to use standard Metanodes to clean up your workflow and cluster common subparts until you actually notice their limits.

Task

Undo the packaging. First select the **(Wrapped) Metanode**, open the context menu (right-click) and select **(Wrapped) Metanode > Expand**.

Advanced topic: R integration

KNIME provides a large number of nodes for a wide range of statistical analysis, machine learning, data processing, and visualization. Still, more recent statistical analysis methods, specialized visualizations or cutting edge algorithms may not be covered in KNIME. In order to expand its capabilities beyond the readily available nodes, external scripting languages can be integrated. In this tutorial, we primarily use scripts of the powerful statistical computing language R. Note that this part is considered advanced and might be difficult to follow if you are not familiar with R. In this case you might skip this part.

R View (Table) allows to seamlessly include R scripts into KNIME. We will demonstrate on a minimal example how such a script is integrated.

Task

First we need some example data in KNIME, which we will generate using the **Data Generator** node. You can keep the default settings and execute the node. The table contains four columns, each containing random coordinates and one column containing a cluster number (Cluster_0 to Cluster_3). Now place a **R View (Table)** node into the workflow and connect the upper output port of the **Data Generator** node to the input of the **R View (Table)** node. Right-click and configure the node. If you get an error message like `Execute failed: R_HOME does not contain a folder with name 'bin'.` or `Execution failed: R Home is invalid.: please change the R settings in the preferences.` To do so open **File > Preferences > KNIME > R** and enter the path to your R installation (the folder that contains the bin directory. (e.g., C:\Program Files\RR-3.4.3)).

If you get an error message like: `"Execute failed: Could not find Rserve package. Please install it in your R installation by running "install.packages('Rserve')"."` You may need to run your R binary as administrator (In windows explorer: right-click "Run as administrator") and enter `install.packages('Rserve')` to install the package.

If R is correctly recognized we can start writing an R script. Consider that we are interested in plotting the first and second coordinates and color them according to their cluster number. In R this can be done in a single line. In the **R view (Table)** text editor, enter the following code:

```
plot(x=knime.in$Universe_0_0, y=knime.in$Universe_0_1, main="Plotting column Universe_0_
↪ 0 vs. Universe_0_1", col=knime.in$"Cluster Membership")
```

Explanation: The table provided as input to the **R View (Table)** node is available as **R data.frame** with name `knime.in`. Columns (also listed on the left side of the R View window) can be accessed in the usual R way by first specifying the

`data.frame` name and then the column name (e.g., `knime.in$Universe_0_0`). `plot` is the plotting function we use to generate the image. We tell it to use the data in column `Universe_0_0` of the dataframe object `knime.in` (denoted as `knime.in$Universe_0_0`) as x-coordinate and the other column `knime.in$Universe_0_1` as y-coordinate in the plot. `main` is simply the main title of the plot and `col` the column that is used to determine the color (in this case it is the `Cluster Membership` column).

Now press the Eval script and Show plot buttons.

Note: Note that we needed to put some extra quotes around `Cluster Membership`. If we omit those, R would interpret the column name only up to the first space (`knime.in$Cluster`) which is not present in the table and leads to an error. Quotes are regularly needed if column names contain spaces, tabs or other special characters like `$` itself.

Label-free quantification of peptides

Introduction

In the following 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 of the last chapter and, step-by-step, build a label-free quantification workflow.

Peptide identification

As a start, we will extend the minimal workflow so that it performs a peptide identification using the OMSSA⁹ 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.

Let's start by replacing the input files in our **Input Files** node by the three mzML files in **Example Data > Labelfree > datasets > lfqxspikeinxdilutionx1-3.mzML**. This is a reduced toy dataset where each of the three runs contains a constant background of *S. pyogenes* peptides as well as human spike-in peptides in different concentrations.¹⁰

- 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 `Example_DataLabelfreedatabasesess_pyo_sf370_potato_human_target_decoy_with_contaminants.fasta`, and we have a very basic peptide identification workflow.

Note: You might also want to save your new identification workflow under a different name. Have a look at duplicating workflows for information on how to create copies of workflows.

⁹ 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). 30

¹⁰ A. Chawade, M. Sandin, J. Teleman, J. Malmström, and F. Levander, Data Processing Has Major Impact on the Outcome of Quantitative Label-Free LC-MS Analysis, *Journal of Proteome Research* 14(2), 676–687 (2015), PMID: 25407311, arXiv:<http://dx.doi.org/10.1021/pr500665j>, doi:10.1021/pr500665j. 30

- 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 an mzML file in **TOPPView**.
- Here, annotate this spectrum 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**. All identified peptides can be seen using this view. User can also browse the corresponding MS2 spectra.

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

- The search results stored in the idXML file can also be read back into a KNIME table for inspection and subsequent analyses: Add a **TextExporter** node from **Community Nodes > OpenMS > File Handling** to your workflow and connect the output port of your **OMSSAAdapter** (the same port **ZipLoopEnd** is connected to) to its input port. This tool will convert the idXML file to a more human-readable text file which can also be read into a KNIME table using the **IDTextReader** node. Add an **IDTextReader** node (**Community Nodes > OpenMS > Conversion**) after **TextExporter** and execute it. Now you can right click **IDTextReader** and select **ID Table** to browse your peptide identifications.
- From here, you can use all the tools KNIME offers for analyzing the data in this table. As a simple example, add a **Histogram (local)** node (from category **Views - Local (Swing)**) node after **IDTextReader**, double-click it, select **peptide_charge** as Histogram column, hit **OK**, and execute it. Right-clicking and selecting **Interactive View: Histogram view** will open a plot showing the charge state distribution of your identifications.

In the next step, we will tweak the parameters of OMSSA to better reflect the instrument's accuracy. 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**. The dataset was recorded using an LTQ Orbitrap XL mass spectrometer, set the precursor mass tolerance to a smaller value, say 5 ppm. Set **precursor_mass_tolerance** to 5 and **precursor_error_units** to ppm.

Note: Whenever you change the configuration of a node, the node as well as all its successors will be reset to the Configured state (all node results are discarded and need to be recalculated by executing the nodes again).

- Set **max_precursor_charge** to 5, in order to also search for peptides with charges up to 5.
- 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 to open the drop down list where you can select the the correct modification.

- A common step in analysis 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. target versus decoy), we have to index the results. To this end, 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 (see below figure).

Tip: 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 “DECOY_”, so we have to set `decoy_string` to `DECOY_` and `decoy_string_position` to `prefix` in the configuration dialog of **PeptideIndexer**.
- Now we can go for the FDR estimation, which the **FalseDiscoveryRate** node will calculate for us (you will find it in **Community Nodes > OpenMS > ID Processing**).
- In order to set the FDR level to 1%, we need an **IDFilter** node from **Community Nodes > OpenMS > ID Processing**. Configuring its parameter `score→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 **IDTextReader** like you did before. 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 Metanode. For this, select all nodes inside the **ZipLoop** (including the **Input File** node) and right-click to select **Collapse into Metanode** and name it **ID**. Metanodes are useful when you construct even larger workflows and want to keep an overview.

The below images shows OMSSA ID pipeline including FDR filtering.

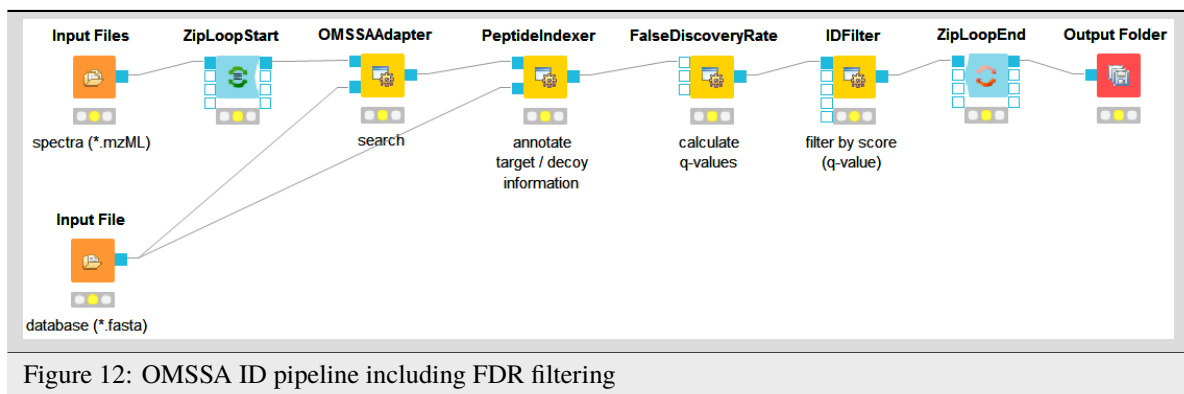


Figure 12: OMSSA ID pipeline including FDR filtering

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. Otherwise, just continue with quantification.

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** and a **XTandemAdapter** **Community Nodes > OpenMS > Identification** node and set its parameters and ports analogously to the **OMSSAAdapter**. In **XTandem**, to get more evenly distributed scores, we decrease the number of candidates a bit by setting the precursor mass tolerance to 5 ppm and the fragment mass tolerance to 0.1 Da.

- 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 GenericKnode > 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 for the tutorial files, you have to set **PeptideIndexer's enzyme→specificity** parameter to **none** to accept X!Tandem's non-tryptic identifications as well.

In the end, the ID processing part of the workflow can be collapsed into a Metanode to keep the structure clean (see below figure which shows complete consensus identification workflow).

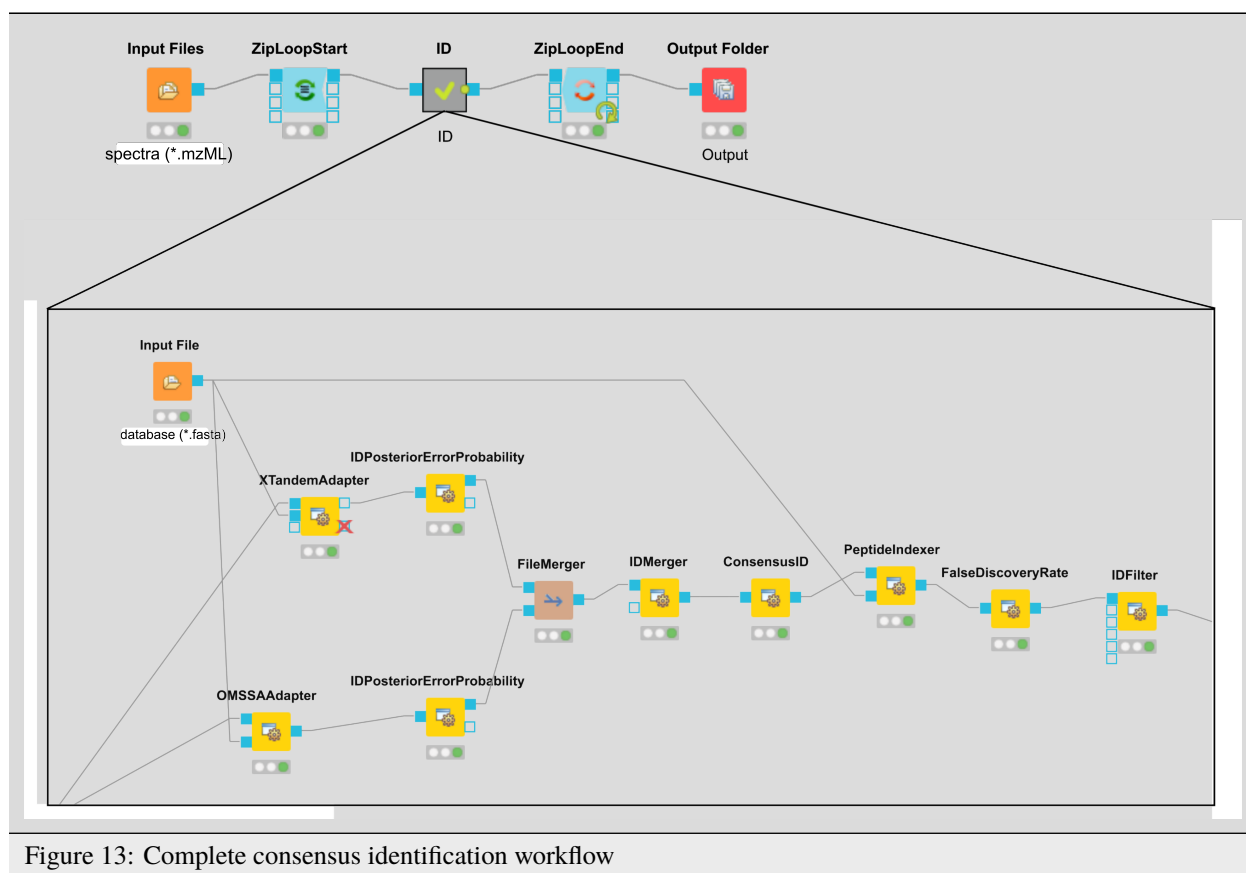


Figure 13: Complete consensus identification workflow

Quantification

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

- Add a **FeatureFinderCentroided** node from **Community Nodes > OpenMS > Quantitation** which gets input from the first output port of the **ZipLoopStart** node. Also, add an **IDMapper** node (from **Community Nodes > OpenMS > ID Processing**) which receives input from the **FeatureFinderCentroided** node (Port 1) and the ID Metanode (or **IDFilter** node (Port 0) if you haven't used the Metanode). The output of the **IDMapper** node is then connected to an in port of 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 overall intensity.
- **FeatureFinderCentroided** produces a featureXML file as output, 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. Open the mzML file of your data to display the raw peak intensities.
- 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. To this end, open the featureXML file in TOPPView by clicking on File Open file and add it to a new layer (Open in New layer). The features are now visualized on top of your raw data. If you zoom in on a small region, you should be able to see the individual boxes around features that have been detected (see Fig. 14). If you hover over the the feature centroid (small circle indicating the chromatographic apex of monoisotopic trace) additional information of the feature is displayed.

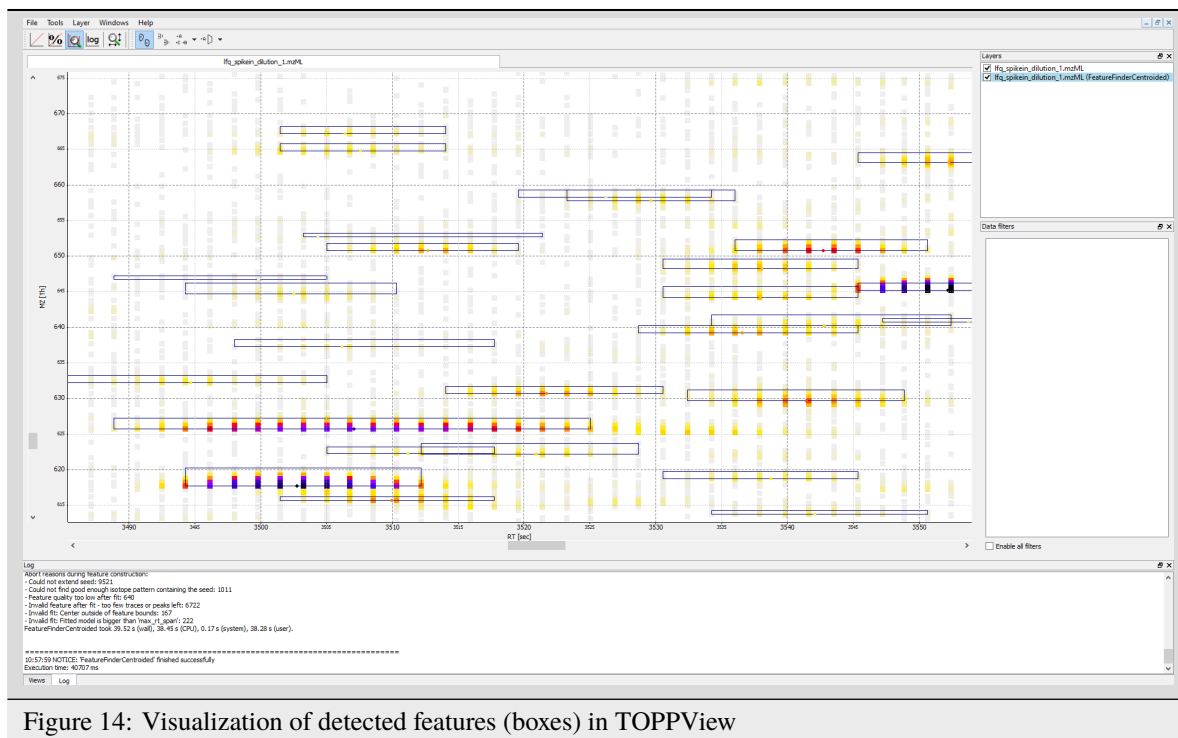


Figure 14: Visualization of detected features (boxes) in TOPPView

Note: The chromatographic RT range of a feature is about 30-60 s and its m/z range around 2.5 m/z in this

dataset. If you have trouble zooming in on a feature, select the full RT range and zoom only into the m/z dimension by holding down Ctrl (cmd on macOS) 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**.

The following image shows the final constructed workflow:

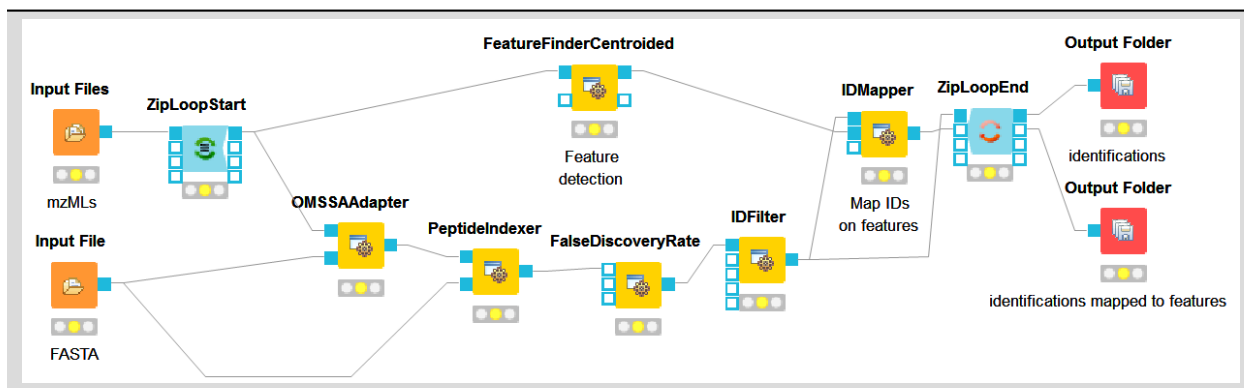


Figure 15: Extended workflow featuring peptide identification and quantification

Combining quantitative information across several label-free experiments

So far, we successfully performed peptide identification as well as quantification on individual LC-MS runs. For differential label-free analyses, however, we need to identify and 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 and extend it a bit further, so that it is able to find and link features across several runs.

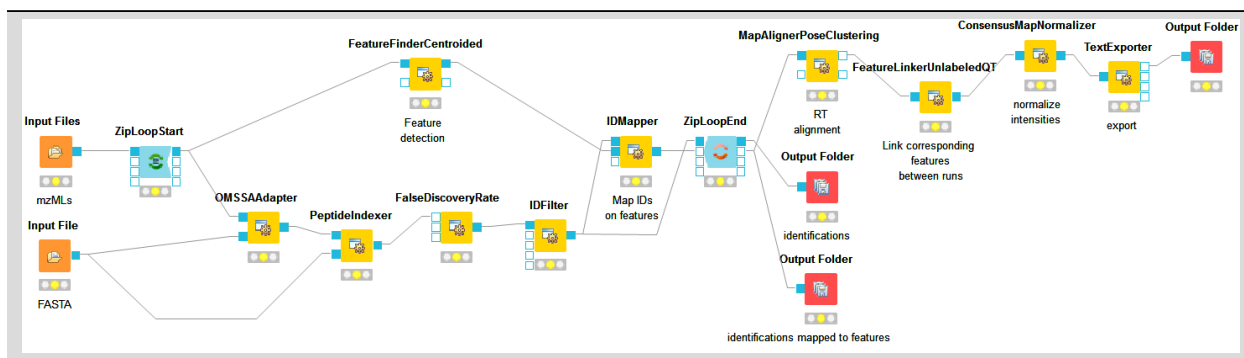


Figure 16: Complete identification and label-free quantification workflow

- 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** node in **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 that **OutputTypes** is set to **featureXML**.

- With the **FeatureLinkerUnlabeledQT** node in **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** node in **Community Node > 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, export the resulting normalized consensusXML file to a csv format using the **TextExporter** node. 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 also be opened with external tools, e.g., Microsoft Excel, for downstream statistical analyses.

Basic data analysis in KNIME

For downstream analysis of the quantification results within the KNIME environment, you can use the **Consensus-TextReader** node in **Community Nodes > OpenMS > Conversion** instead of the **Output Folder** node to convert the output into a KNIME table (indicated by a triangle as output port). After running the node you can view the KNIME table by right-clicking on the **ConsensusTextReader** node and selecting **Consensus Table**. Every row in this table corresponds to a so-called consensus feature, i.e., a peptide signal quantified across several runs. The first couple of columns describe the consensus feature as a whole (average RT and m/z across the maps, charge, etc.). The remaining columns describe the exact positions and intensities of the quantified features separately for all input samples (e.g., **intensity_0** is the intensity of the feature in the first input file). The last 11 columns contain information on peptide identification.

- Now, let’s say we want to plot the log intensity distributions of the human spike-in peptides for all input files. In addition, we will plot the intensity distributions of the background peptides.
- As shown in Fig. 17, add a **Row Splitter** node (**Data Manipulation > Row > Filter**) after the **ConsensusTextReader** node. Double-click it to configure. The human spike-in peptides have accessions starting with “hum”. Thus, set the column to apply the test to: **accessions**, select **pattern matching** as matching criterion, enter **hum** into the corresponding text field, and check the **contains wild cards** box. Press **OK** and execute the node.
- **Row Splitter** produces two output tables: the first one contains all rows from the input table matching the filter criterion, and the second table contains all other rows. You can inspect the tables by right-clicking and selecting **Filtered** and **Filtered Out**. The former table should now only contain peptides with a human accession, whereas the latter should contain all remaining peptides (including unidentified ones).
- Now, since we only want to plot intensities, we can add a **Column Filter** node by going to **Data Manipulation > Column Filter**. Connect its input port to the **Filtered output** port of the **Row Filter** node, and open its configuration dialog. We could either manually select the columns we want to keep, or, more elegantly, select **Wildcard/Regex Selection** and enter **intensity_?** as the pattern. KNIME will interactively show you which columns your pattern applies to while you’re typing.
- Since we want to plot log intensities, we will now compute the log of all intensity values in our table. The easiest way to do this in KNIME is a small piece of R code. Add an **R Snippet** node R after **Column Filter** and double-click to configure. In the R Script text editor, enter the following code:

```

x <- knime.in      # store copy of input table in x

x[x == 0] <- NA    # replace all zeros by NA (= missing value)

x <- log10(x)      # compute log of all values
knime.out <- x     # write result to output table

```

- Now we are ready to plot! Add a **Box Plot (local)** node Views -Swing (local) after the **R Snippet** node, execute it, and open its view. If everything went well, you should see a significant fold change of your human peptide intensities across the three runs.
- To verify that the concentration of background peptides is constant in all three runs, copy and paste the three nodes after **Row Splitter** and connect the duplicated **Column Filter** to the second output port (Filtered Out) of **Row Splitter**, as shown in Fig. 17. Execute and open the view of your second **Box Plot**.

You have now constructed an entire identification and label-free quantification workflow including a simple data analysis using KNIME. The final workflow should like the workflow shown in the following image:

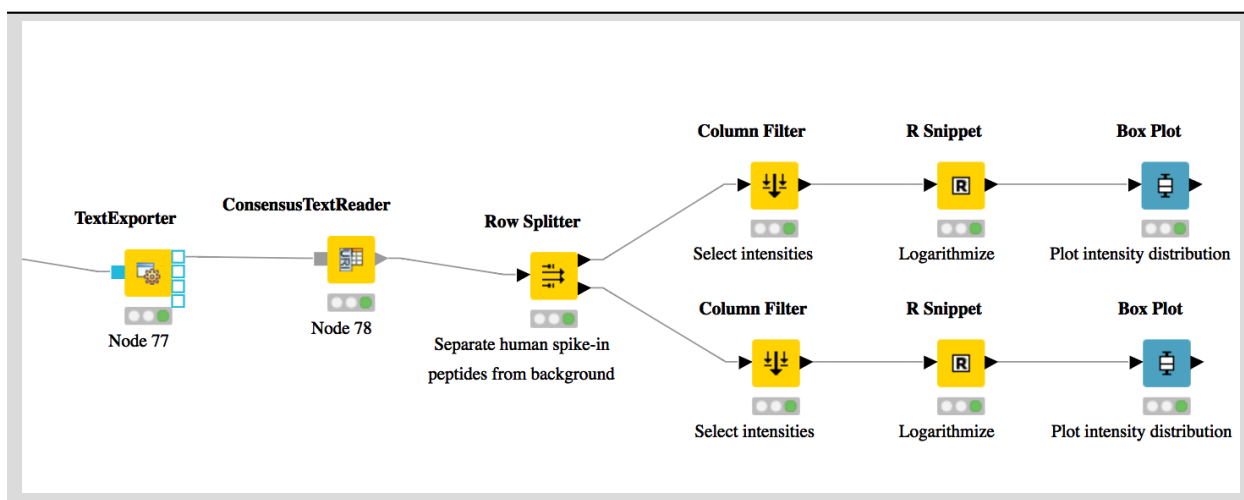


Figure 17: Simple KNIME data analysis example for LFQ

Identification and quantification of the iPRG2015 data with subsequent MSstats analysis

Advanced downstream data analysis of quantitative mass spectrometry-based proteomics data can be performed using MSstats¹¹. This tool can be combined with an OpenMS preprocessing pipeline (e.g. in KNIME). The OpenMS experimental design is used to present the data in an MSstats-conformant way for the analysis. Here, we give an example how to utilize these resources when working with quantitative label-free data. We describe how to use OpenMS and MSstats for the analysis of the ABRF iPRG2015 dataset¹².

Note: Reanalysing the full dataset from scratch would take too long. In the following tutorial, we will focus on just the conversion process and the downstream analysis.

¹¹ A. Chawade, M. Sandin, J. Teleman, J. Malmström, and F. Levander, Data Processing Has Major Impact on the Outcome of Quantitative Label-Free LC-MS Analysis, Journal of Proteome Research 14(2), 676–687 (2015), PMID: 25407311, arXiv:<http://dx.doi.org/10.1021/pr500665j>, doi:10.1021/pr500665j. 30

¹² M. Choi, Z. F. Eren-Dogu, C. Colangelo, J. Cottrell, M. R. Hoopmann, E. A. Kapp, S. Kim, H. Lam, T. A. Neubert, M. Palmblad, B. S. Phinney, S. T. Weintraub, B. MacLean, and O. Vitek, ABRF Proteome Informatics Research Group (iPRG) 2015 Study: Detection of Differentially Abundant Proteins in Label-Free Quantitative LC-MS/MS Experiments, J. Proteome Res. 16(2), 945–957 (2017), doi: 10.1021/acs.jproteome.6b00881. 40

Excursion MSstats

The R package **MSstats** can be used for statistical relative quantification of proteins and peptides in mass spectrometry-based proteomics. Supported are label-free as well as labeled experiments in combination with data-dependent, targeted and data independent acquisition. Inputs can be identified and quantified entities (peptides or proteins) and the output is a list of differentially abundant entities, or summaries of their relative abundance. It depends on accurate feature detection, identification and quantification which can be performed e.g. by an OpenMS workflow. MSstats can be used for data processing & visualization, as well as statistical modeling & inference. Please see [Page 82, 11](#) and the [MSstats](#) website for further information.

Dataset

The iPRG (Proteome Informatics Research Group) dataset from the study in 2015, as described in [Page 82, 12](#), aims at evaluating the effect of statistical analysis software on the accuracy of results on a proteomics label-free quantification experiment. The data is based on four artificial samples with known composition (background: 200 ng *S. cerevisiae*). These were spiked with different quantities of individual digested proteins, whose identifiers were masked for the competition as yeast proteins in the provided database (see Table 1).

Identification and quantification

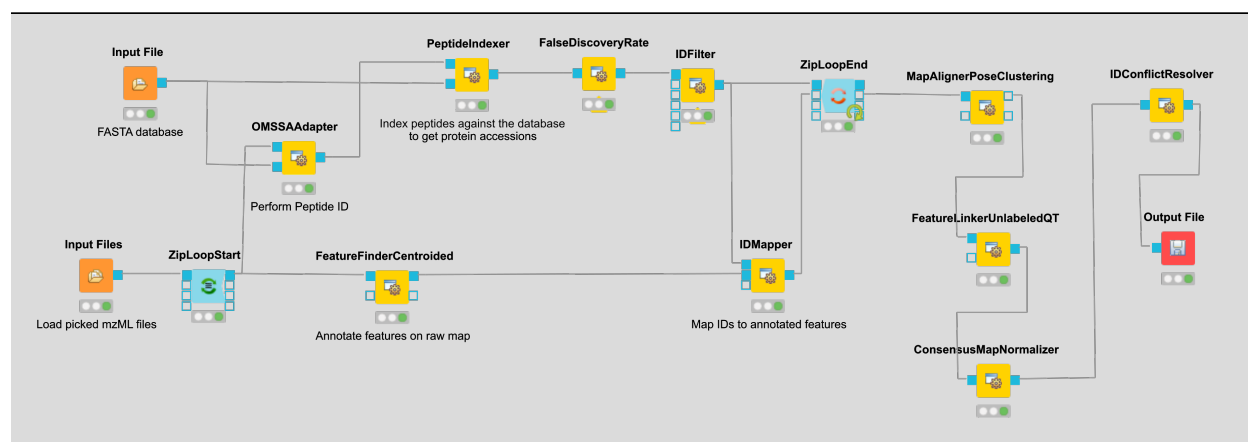


Figure 18: KNIME data analysis of iPRG LFQ data.

The iPRG LFQ workflow (Fig. 18) consists of an identification and a quantification part. The identification is achieved by searching the computationally calculated MS2 spectra from a sequence database (**Input File** node, here with the given database from iPRG: `ExampleData\iPRG2015\database\iPRG2015\target\decoy\contaminants.fasta` against the MS2 from the original data (**Input Files** node with all mzMLs following `ExampleData\iPRG2015\datasets\JD06232014\sample*.mzML` using the **OMSSAAAdapter**.

Note: If you want to reproduce the results at home, you have to download the iPRG data in mzML format and perform peak picking on it or convert and pick the raw data with `msconvert`.

Afterwards, the results are scored using the **FalseDiscoveryRate** node and filtered to obtain only unique peptides (**IDFilter**) since MSstats does not support shared peptides, yet. The quantification is achieved by using the **FeatureFinderCentroided** node, which performs the feature detection on the samples (maps). In the end the quantification

results are combined with the filtered identification results (**IDMapper**). In addition, a linear retention time alignment is performed (**MapAlignerPoseClustering**), followed by the feature linking process (**FeatureLinkerUnlabeledQT**). The **ConsensusMapNormalizers** is used to normalize the intensities via robust regression over a set of maps and the **IDConflictResolver** assures that only one identification (best score) is associated with a feature. The output of this workflow is a consensusXML file, which can now be converted using the **MSstatsConverter** (see Conversion and downstream analysis section).

Experimental design

As mentioned before, the downstream analysis can be performed using MSstats. In this case, an experimental design has to be specified for the OpenMS workflow. The structure of the experimental design used in OpenMS in case of the iPRG dataset is specified in Table 2.

An explanation of the variables can be found in Table 3.

The conditions are highly dependent on the type of experiment and on which kind of analysis you want to perform. For the MSstats analysis the information which sample belongs to which condition and if there are biological replicates are mandatory. This can be specified in further condition columns as explained in Table 3. For a detailed description of the MSstats-specific terminology, see their documentation e.g. in the R vignette.

Conversion and downstream analysis

Conversion of the OpenMS-internal consensusXML format (which is an aggregation of quantified and possibly identified features across several MS-maps) to a table (in MSstats-conformant CSV format) is very easy. First, create a new KNIME workflow. Then, run the **MSstatsConverter** node with a consensusXML and the manually created (e.g. in Excel) experimental design as inputs (loaded via **Input File** nodes). The first input can be found in:

ExampleData\iPRG2015\openmsLFQResults\iPRG\lfq.consensusXML

This file was generated by using the Workflows\openmsLFQiPRG2015.knwf workflow (seen in Fig. 18). The second input is specified in:

ExampleData\iPRG2015\experimentalDesign.tsv

Adjust the parameters in the config dialog of the converter to match the given experimental design file and to use a simple summing for peptides that elute in multiple features (with the same charge state, i.e. m/z value).

parameter	value
<i>msstats_bioreplicate</i>	MSstats_Bioreplicate
<i>msstats_condition</i>	MSstats_Condition
<i>labeled_reference_peptides</i>	false
<i>retention_time_summarization_method (advanced)</i>	sum

The downstream analysis of the peptide ions with MSstats is performed in several steps. These steps are reflected by several KNIME R nodes, which consume the output of **MSstatsConverter**. The outline of the workflow is shown in Figure 19.

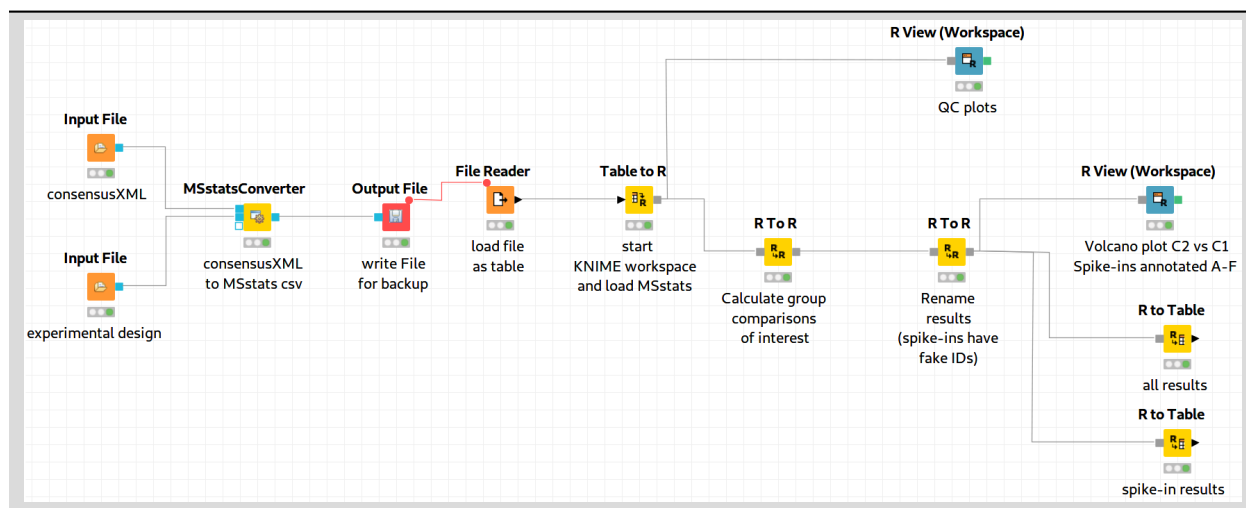


Figure 19: MSstats analysis using KNIME. The individual steps (Preprocessing, Group Comparisons, Result Data Renaming, and Export) are split among several consecutive nodes.

We load the file resulting from **MSstatsConverter** either by saving it with an **Output File** node and reloading it with the **File Reader**. Alternatively, for advanced users, you can use a URI Port to Variable node and use the variable in the File Reader config dialog (V button - located on the right of the **Browse** button) to read from the temporary file.

Preprocessing

The first node (**Table to R**) loads MSstats as well as the data from the previous KNIME node and performs a pre-processing step on the input data. The following inline R script (needs to be pasted into the config dialog of the node):

```
library(MSstats)
data <- knime.in
quant <- OpenMStoMSstatsFormat(data, removeProtein_with1Feature = FALSE)
```

The inline R script allows further preparation of the data produced by **MSstatsConverter** before the actual analysis is performed. In this example, the lines with proteins, which were identified with only one feature, were retained. Alternatively they could be removed. In the same node, most importantly, the following line transforms the data into a format that is understood by MSstats:

```
processed.quant <- dataProcess(quant, censoredInt = 'NA')
```

Here, **dataProcess** is one of the most important functions that the R package provides. The function performs the following steps:

1. Logarithm transformation of the intensities
2. Normalization
3. Feature selection
4. Missing value imputation
5. Run-level summarization

In this example, we just state that missing intensity values are represented by the NA string.

Group Comparison

The goal of the analysis is the determination of differentially-expressed proteins among the different conditions C1-C4. We can specify the comparisons that we want to make in a *comparison* matrix. For this, let's consider the following example:

This matrix has the following properties:

- The number of rows equals the number of comparisons that we want to perform, the number of columns equals the number of conditions (here, column 1 refers to C1, column 2 to C2 and so forth).
- The entries of each row consist of exactly one 1 and one -1, the others must be 0.
- The condition with the entry 1 constitutes the enumerator of the log2 fold-change. The one with entry -1 denotes the denominator. Hence, the first row states that we want calculate:

$$\log_2 \frac{C_2}{C_1}(1.6)$$

We can generate such a matrix in R using the following code snippet in (for example) a new **R to R** node that takes over the R workspace from the previous node with all its variables:

```
comparison1<-matrix(c(-1,1,0,0),nrow=1)
comparison2<-matrix(c(-1,0,1,0),nrow=1)

comparison3<-matrix(c(-1,0,0,1),nrow=1)
comparison4<-matrix(c(0,-1,1,0),nrow=1)

comparison5<-matrix(c(0,-1,0,1),nrow=1)
comparison6<-matrix(c(0,0,-1,1),nrow=1)

comparison <- rbind(comparison1, comparison2, comparison3, comparison4, comparison5,↵
↵comparison6)
row.names(comparison)<-c("C2-C1", "C3-C1", "C4-C1", "C3-C2", "C4-C2", "C4-C3")
```

Here, we assemble each row in turn, concatenate them at the end, and provide row names for labeling the rows with the respective condition. In MSstats, the group comparison is then performed with the following line:

```
test.MSstats <- groupComparison(contrast.matrix=comparison, data=processed.quant)
```

No more parameters need to be set for performing the comparison.

Result processing

In a next R to R node, the results are being processed. The following code snippet will rename the spiked-in proteins to A,B,C,D,E, and F and remove the names of other proteins, which will be beneficial for the subsequent visualization, as for example performed in Figure 20:

```
test.MSstats.cr <- test.MSstats$ComparisonResult

# Rename spiked ins to A,B,C....
pnames <- c("A", "B", "C", "D", "E", "F")

names(pnames) <- c(
  "sp|P44015|VAC2_YEAST",
  "sp|P55752|ISCB_YEAST",

  "sp|P44374|SFG2_YEAST",
  "sp|P44983|UTR6_YEAST",
  "sp|P44683|PGA4_YEAST",

  "sp|P55249|ZRT4_YEAST"
)

test.MSstats.cr.spikedins <- bind_rows(
  test.MSstats.cr[grep("P44015", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P55752", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P44374", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P44683", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P44983", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P55249", test.MSstats.cr$Protein),]
)
# Rename Proteins

test.MSstats.cr.spikedins$Protein <- sapply(test.MSstats.cr.spikedins$Protein, function(x) {pnames[as.character(x)]})

test.MSstats.cr$Protein <- sapply(test.MSstats.cr$Protein, function(x) {

  x <- as.character(x)

  if (x %in% names(pnames)) {

    return(pnames[as.character(x)])
```

(continues on next page)

(continued from previous page)

```

    } else {
      return("")
    }
  })

```

Export

The last four nodes, each connected and making use of the same workspace from the last node, will export the results to a textual representation and volcano plots for further inspection. Firstly, quality control can be performed with the following snippet:

```

qcplot <- dataProcessPlots(processed.quant, type="QCplot",
  ylimDown=0,

which.Protein = 'allonly',
width=7, height=7, address=F)

```

The code for this snippet is embedded in the first output node of the workflow. The resulting boxplots show the log2 intensity distribution across the MS runs. The second node is an **R View (Workspace)** node that returns a Volcano plot which displays differentially expressed proteins between conditions C2 vs. C1. The plot is described in more detail in the following Result section. This is how you generate it:

```

groupComparisonPlots(data=test.MSstats.cr, type="VolcanoPlot",

  width=12, height=12,dot.size = 2,ylimUp = 7,

  which.Comparison = "C2-C1",
  address=F)

```

The last two nodes export the MSstats results as a KNIME table for potential further analysis or for writing it to a (e.g. csv) file. Note that you could also write output inside the Rscript if you are familiar with it. Use the following for an **R to Table** node exporting all results:

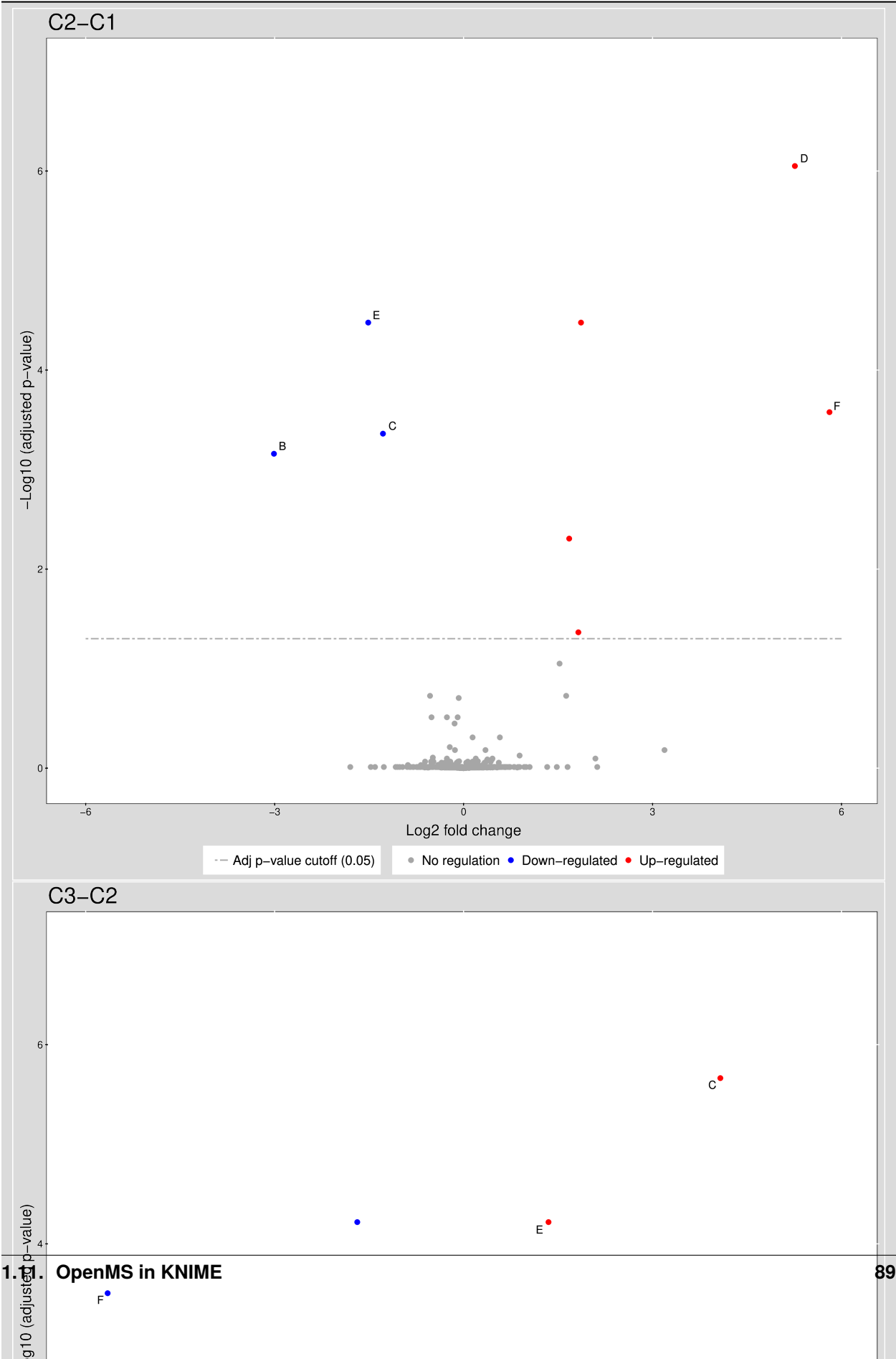
```
knime.out <- test.MSstats.cr
```

And this for an **R to Table** node exporting only results for the spike-ins:

```
knime.out <- test.MSstats.cr.spikedins
```

Result

An excerpt of the main result of the group comparison can be seen in Figure 20.



The Volcano plots show differently expressed spiked-in proteins. In the left plot, which shows the fold-change C2-C1, we can see the proteins D and F (sp|P44983|UTR6_YEAST and sp|P55249|ZRT4_YEAST) are significantly over-expressed in C2, while the proteins B,C, and E (sp|P55752|ISCB_YEAST, sp|P55752|ISCB_YEAST, and sp|P44683|PGA4_YEAST) are under-expressed. In the right plot, which shows the fold-change ratio of C3 vs. C2, we can see the proteins E and C (sp|P44683|PGA4_YEAST and sp|P44374|SFG2_YEAST) over-expressed and the proteins A and F (sp|P44015|VAC2_YEAST and sp|P55249|ZRT4_YEAST) under-expressed. The plots also show further differentially-expressed proteins, which do not belong to the spiked-in proteins.

The full analysis workflow can be found under: WorkflowsMSstatsstatPostProcessingiPRG2015.knwf

Protein inference

In the last chapter, we have successfully quantified peptides in a label-free experiment. As a next step, we will further extend this label-free quantification workflow by protein inference and protein quantification capabilities. This workflow uses some of the more advanced concepts of KNIME, as well as a few more nodes containing R code. For these reasons, you will not have to build it yourself. Instead, we have already prepared and copied this workflow to the USB sticks. Just import Workflowslabelfree_with_protein_quantification.knwf into KNIME via the menu entry **File > Import KNIME workflow > Select file** and double-click the imported workflow in order to open it.

Before you can execute the workflow, you again have to correct the locations of the files in the Input Files nodes (don't forget the one for the FASTA database inside the "ID" meta node). Try and run your workflow by executing all nodes at once.

Extending the LFQ workflow by protein inference and quantification

We have made the following changes compared to the original label-free quantification workflow from the last chapter:

- First, we have added a **ProteinQuantifier** node and connected its input port to the output port of the **ConsensusMapNormalizer** node.
- This already enables protein quantification. **ProteinQuantifier** quantifies peptides by summarizing over all observed charge states and proteins by summarizing over their quantified peptides. It stores two output files, one for the quantified peptides and one for the proteins.
- In this example, we consider only the protein quantification output file, which is written to the first output port of **ProteinQuantifier**.
- Because there is no dedicated node in KNIME to read back the **ProteinQuantifier** output file format into a KNIME table, we have to use a workaround. Here, we have added an additional URI Port to Variable node which converts the name of the output file to a so-called "flow variable" in KNIME. This variable is passed on to the next node **CSV Reader**, where it is used to specify the name of the input file to be read. If you double-click on **CSV Reader**, you will see that the text field, where you usually enter the location of the CSV file to be read, is greyed out. Instead, the flow variable is used to specify the location, as indicated by the small green button with the "v=?" label on the right.
- The table containing the **ProteinQuantifier** results is filtered one more time in order to remove decoy proteins. You can have a look at the final list of quantified protein groups by right-clicking the **Row Filter** and selecting **Filtered**.
- By default, i.e., when the second input port **protein_groups** is not used, **ProteinQuantifier** quantifies proteins using only the unique peptides, which usually results in rather low numbers of quantified proteins.
- In this example, however, we have performed protein inference using Fido and used the resulting protein grouping information to also quantify indistinguishable proteins. In fact, we also used a greedy method in **FidoAdapter** (parameter **greedy_group_resolution**) to uniquely assign the peptides of a group to the most probable protein(s) in the respective group. This boosts the number of quantifications but slightly raises the chances to yield distorted protein quantities.

- As a prerequisite for using **FidoAdapter**, we have added an **IDPosteriorErrorProbability** node within the ID meta node, between the **XTandemAdapter** (note the replacement of OMSSA because of ill-calibrated scores) and **PeptideIndexer**. We have set its parameter `prob_correct` to `true`, so it computes posterior probabilities instead of posterior error probabilities (1 - PEP). These are stored in the resulting idXML file and later on used by the Fido algorithm. Also note that we excluded FDR filtering from the standard meta node. Harsh filtering before inference impacts the calibration of the results. Since we filter peptides before quantification though, no potentially random peptides will be included in the results anyway.
- Next, we have added a third outgoing connection to our ID meta node and connected it to the second input port of **ZipLoopEnd**. Thus, KNIME will wait until all input files have been processed by the loop and then pass on the resulting list of idXML files to the subsequent IDMerger node, which merges all identifications from all idXML files into a single idXML file. This is done to get a unique assignment of peptides to proteins over all samples.
- Instead of the meta node **Protein inference with FidoAdapter**, we could have just used a **FidoAdapter** node (**Community Nodes > OpenMS > ID Processing**). However, the meta node contains an additional subworkflow which, besides calling **FidoAdapter**, performs a statistical validation (e.g. (pseudo) receiver operating curves; ROCs) of the protein inference results using some of the more advanced KNIME and R nodes. The meta node also shows how to use **MzTabExporter** and **MzTabReader**.

Statistical validation of protein inference results

In the following section, we will explain the subworkflow contained in the **Protein inference with FidoAdapter** meta node.

Data preparation

For downstream analysis on the protein ID level in KNIME, it is again necessary to convert the idXML-file-format result generated from **FidoAdapter** into a KNIME table.

- We use the **MzTabExporter** to convert the inference results from **FidoAdapter** to a human readable, tab-separated mzTab file. mzTab contains multiple sections, that are all exported by default, if applicable. This file, with its different sections can again be read by the **MzTabReader** that produces one output in KNIME table format (triangle ports) for each section. Some ports might be empty if a section did not exist. Of course, we continue by connecting the downstream nodes with the protein section output (second port).
- Since the protein section contains single proteins as well as protein groups, we filter them for single proteins with the standard **Row Filter**.

ROC curve of protein ID

ROC Curves (Receiver Operating Characteristic curves) are graphical plots that visualize sensitivity (true-positive rate) against fall-out (false positive rate). They are often used to judge the quality of a discrimination method like e.g., peptide or protein identification engines. ROC Curve already provides the functionality of drawing ROC curves for binary classification problems. When configuring this node, select the `opt_global_target_decoy` column as the class (i.e. target outcome) column. We want to find out, how good our inferred protein probability discriminates between them, therefore add `best_search_engine_score[1]` (the inference engine score is treated like a peptide search engine score) to the list of *"Columns containing positive class probabilities"*. View the plot by right-clicking and selecting **View: ROC Curves**. A perfect classifier has an area under the curve (AUC) of 1.0 and its curve touches the upper left of the plot. However, in protein or peptide identification, the ground-truth (i.e., which target identifications are true, which are false) is usually not known. Instead, so called pseudoROC Curves are regularly used to plot the number of target proteins against the false discovery rate (FDR) or its protein-centric counterpart, the q-value. The FDR is approximated by using the target-decoy estimate in order to distinguish true IDs from false IDs by separating target IDs from decoy IDs.

Posterior probability and FDR of protein IDs

ROC curves illustrate the discriminative capability of the scores of IDs. In the case of protein identifications, Fido produces the posterior probability of each protein as the output score. However, a perfect score should not only be highly discriminative (distinguishing true from false IDs), it should also be “calibrated” (for probability indicating that all IDs with reported posterior probability scores of 95% should roughly have a 5% probability of being false. This implies that the estimated number of false positives can be computed as the sum of posterior error probabilities ($= 1 - \text{posterior probability}$) in a set, divided by the number of proteins in the set. Thereby a posterior-probability-estimated FDR is computed which can be compared to the actual target-decoy FDR. We can plot calibration curves to help us visualize the quality of the score (when the score is interpreted as a probability as Fido does), by comparing how similar the target-decoy estimated FDR and the posterior probability estimated FDR are. Good results should show a close correspondence between these two measurements, although a non-correspondence does not necessarily indicate wrong results.

The calculation is done by using a simple R script in R snippet. First, the target decoy protein FDR is computed as the proportion of decoy proteins among all significant protein IDs. Then posterior probabilistic-driven FDR is estimated by the average of the posterior error probability of all significant protein IDs. Since FDR is the property for a group of protein IDs, we can also calculate a local property for each protein: the q-value of a certain protein ID is the minimum FDR of any groups of protein IDs that contain this protein ID. We plot the protein ID results versus two different kinds of FDR estimates in R View(Table) (see Fig. 22).

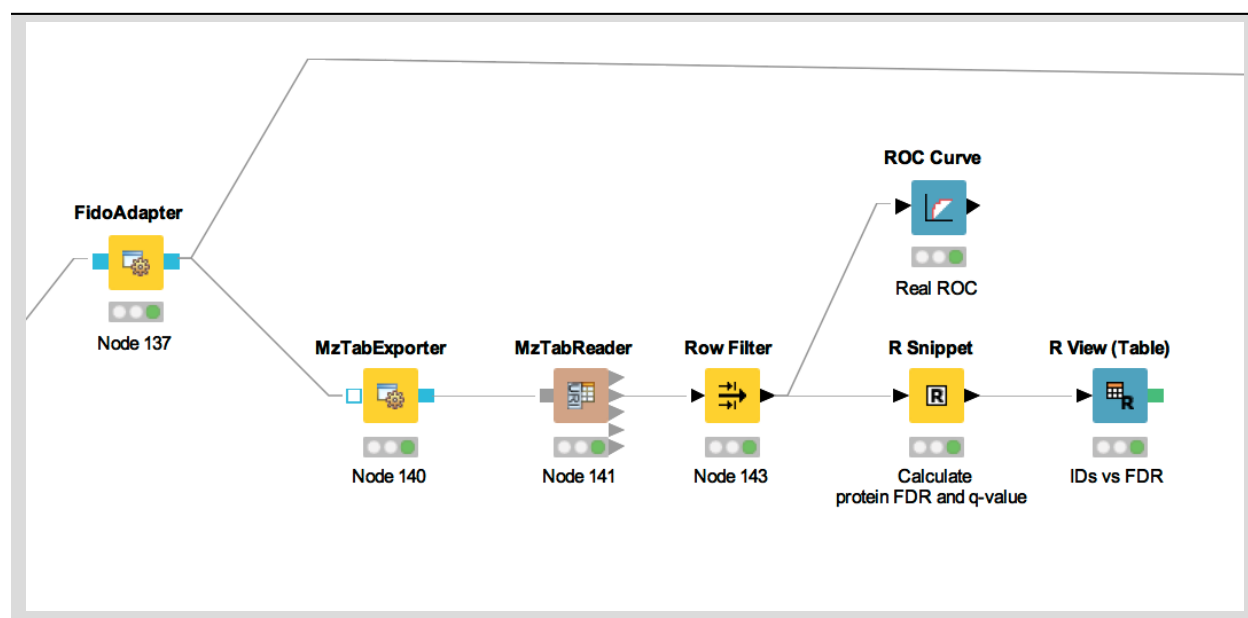


Figure 21: The workflow of statistical analysis of protein inference results

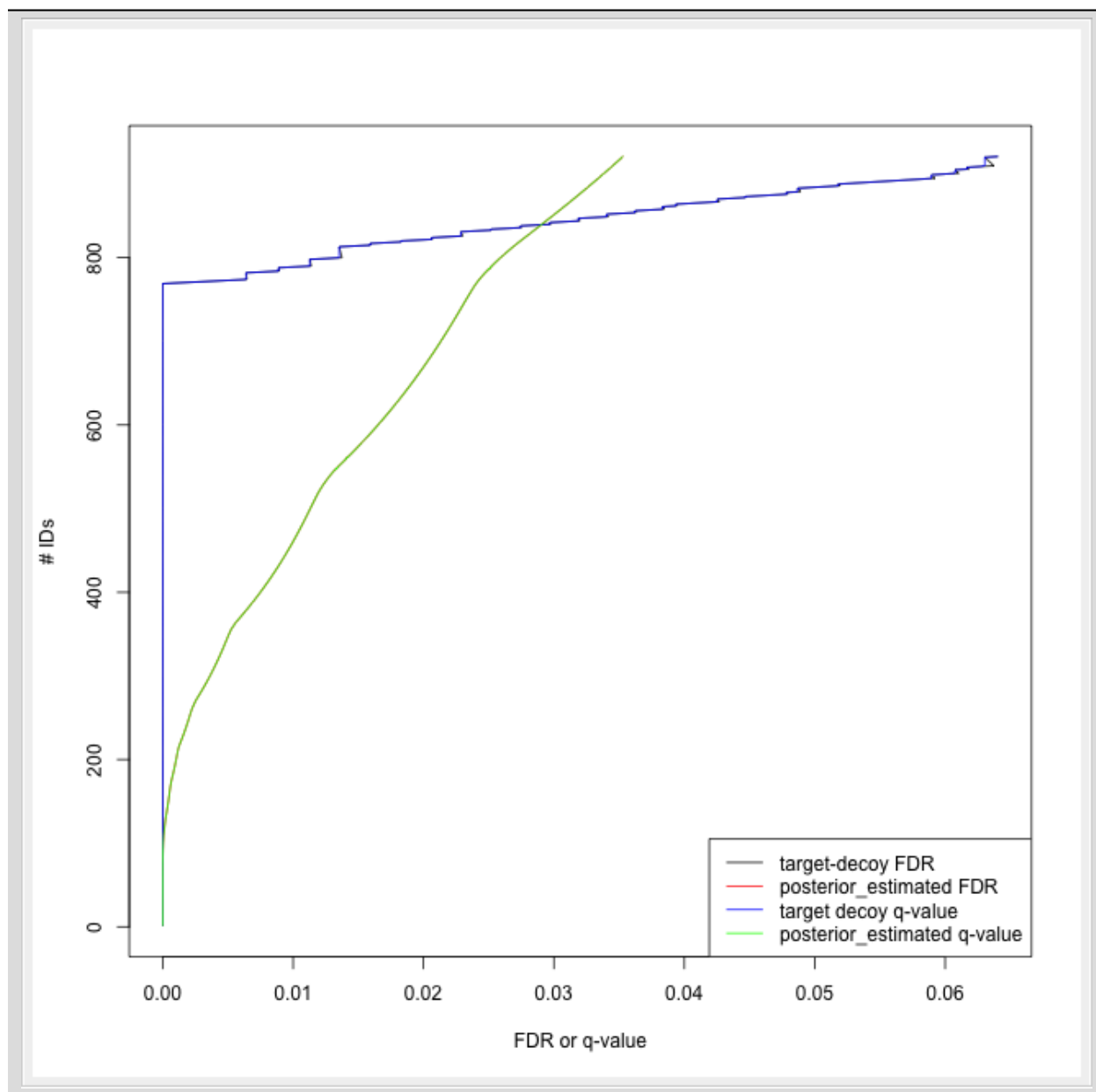


Figure 22: The pseudo-ROC Curve of protein IDs. The accumulated number of protein IDs is plotted on two kinds of scales: target-decoy protein FDR and Fido posterior probability estimated FDR. The largest value of posterior probability estimated FDR is already smaller than 0.04, this is because the posterior probability output from Fido is generally very high

Isobaric analysis

In the last chapters, we identified and quantified peptides in a label-free experiment.

In this section, we would like to introduce a possible workflow for the analysis of isobaric data.

Isobaric analysis workflow

Let's have a look at the workflow (see Fig 23).

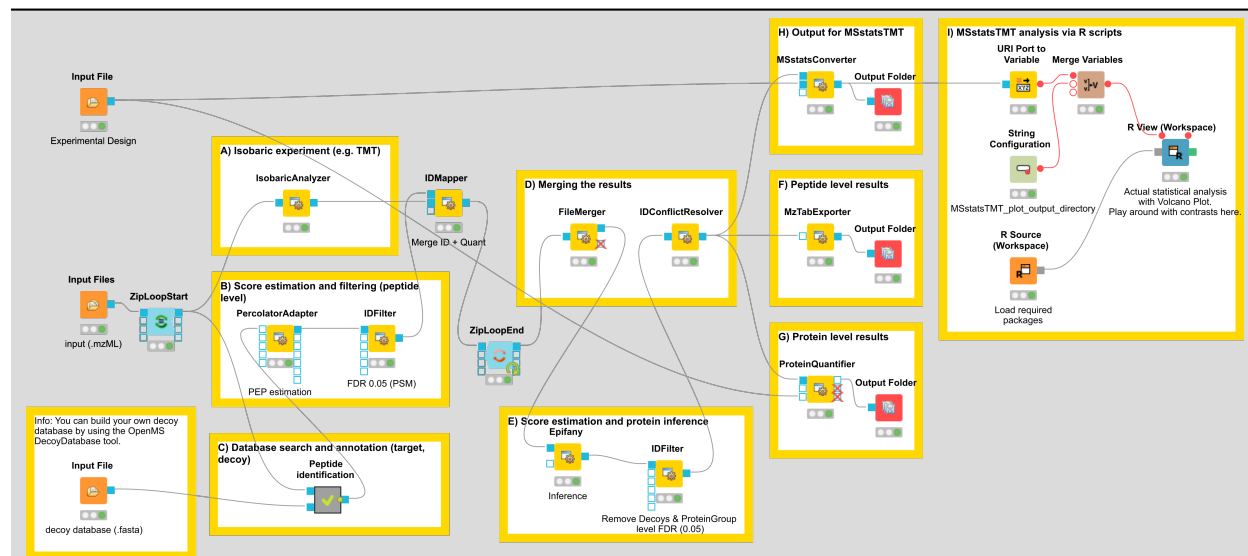


Figure 23: Workflow for the analysis of isobaric data

The full analysis workflow can be found here: [WorkflowsIdentificationquantificationisobaricinferrenceepifanyMSstatsTMT](#)

The workflow has four input nodes. The first for the experimental design to allow for MSstatsTMT compatible export (**MSstatsConverter**). The second for the .mzML files with the centroided spectra from the isobaric labeling experiment and the third one for the .fasta database used for identification. The last one allows to specify an output path for the plots generated by the R View, which runs MSstatsTMT (I). The quantification (A) is performed using the **IsobaricAnalyzer**. The tool is able to extract and normalize quantitative information from TMT and iTRAQ data. The values can be assessed from centroided MS2 or MS3 spectra (if available). Isotope correction is performed based on the specified correction matrix (as provided by the manufacturer). The identification © is applied as known from the previous chapters by using database search and a target-decoy database.

To reduce the complexity of the data for later inference the q-value estimation and FDR filtering is performed on PSM level for each file individually (B). Afterwards the identification (PSM) and quantitative information is combined using the **IDMapper**. After the processing of all available files, the intermediate results are aggregated (**FileMerger** - D). All PSM results are used for score estimation and protein inference (**Epifany**) (E). For detailed information about protein inference please see Chapter 4. Then, decoys are removed and the inference results are filtered via a protein group FDR. Peptide level results can be exported via **MzTabExporter** (F), protein level results can be obtained via the **ProteinQuantifier** (G) or the results can be exported (**MSstatsConverter** - H) and further processed with the following R pipeline to allow for downstream processing using MSstatsTMT.

Please import the workflow from [WorkflowsIdentificationquantificationisobaricinferrenceepifanyMSstatsTMT](#) into KNIME via the menu entry **File > Import KNIME workflow > Select file** and double-click the imported workflow in order to open it. Before you can execute the workflow, you have to correct the locations of the files in the **Input Files**

nodes (don't forget the one for the FASTA database inside the "ID" meta node). Try and run your workflow by executing all nodes at once.

Excursion MSstatsTMT

The R package MSstatsTMT can be used for protein significance analysis in shotgun mass spectrometry-based proteomic experiments with tandem mass tag (TMT) labeling. MSstatsTMT provides functionality for two types of analysis & their visualization: Protein summarization based on peptide quantification and Model-based group comparison to detect significant changes in abundance. It depends on accurate feature detection, identification and quantification which can be performed e.g. by an OpenMS workflow.

In general, MSstatsTMT can be used for data processing & visualization, as well as statistical modeling. Please see¹³ and the MSstats website for further information.

There is also an [online lecture](#) and tutorial for MSstatsTMT from the May Institute Workshop 2020.

Dataset and experimental design

We are using the MSV000084264 ground truth dataset, which consists of TMT10plex controlled mixes of different concentrated UPS1 peptides spiked into SILAC HeLa peptides measured in a dilution series <https://www.omicsdi.org/dataset/massive/MSV000084264>. Figure 24 shows the experimental design. In this experiment, 5 different TMT10plex mixtures – different labeling strategies – were analysed. These were measured in triplicates represented by the 15 MS runs (3 runs each). The example data, database and experimental design to run the workflow can be found [here](#).

TMT10plex reagent		126	127N	127C	128N	128C	129N	129C	130N	130C	131
Mixture 1	Run 1	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 2	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 3	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
Mixture 2	Run 4	Ref	0.5	1	0.667	0.125	1	0.667	0.125	0.5	Ref
	Run 5	Ref	0.5	1	0.667	0.125	1	0.667	0.125	0.5	Ref
	Run 6	Ref	0.5	1	0.667	0.125	1	0.667	0.125	0.5	Ref
Mixture 3	Run 7	Ref	0.125	0.667	1	0.5	0.5	0.125	0.667	1	Ref
	Run 8	Ref	0.125	0.667	1	0.5	0.5	0.125	0.667	1	Ref
	Run 9	Ref	0.125	0.667	1	0.5	0.5	0.125	0.667	1	Ref
Mixture 4	Run 10	Ref	1	0.5	0.125	0.667	0.667	1	0.5	0.125	Ref
	Run 11	Ref	1	0.5	0.125	0.667	0.667	1	0.5	0.125	Ref
	Run 12	Ref	1	0.5	0.125	0.667	0.667	1	0.5	0.125	Ref
Mixture 5	Run 13	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 14	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 15	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref

Figure 24: Experimental Design

The experimental design in table format allows for MSstatsTMT compatible export. The design is represented by two tables. The first one represents the overall structure of the experiment in terms of samples, fractions, labels and

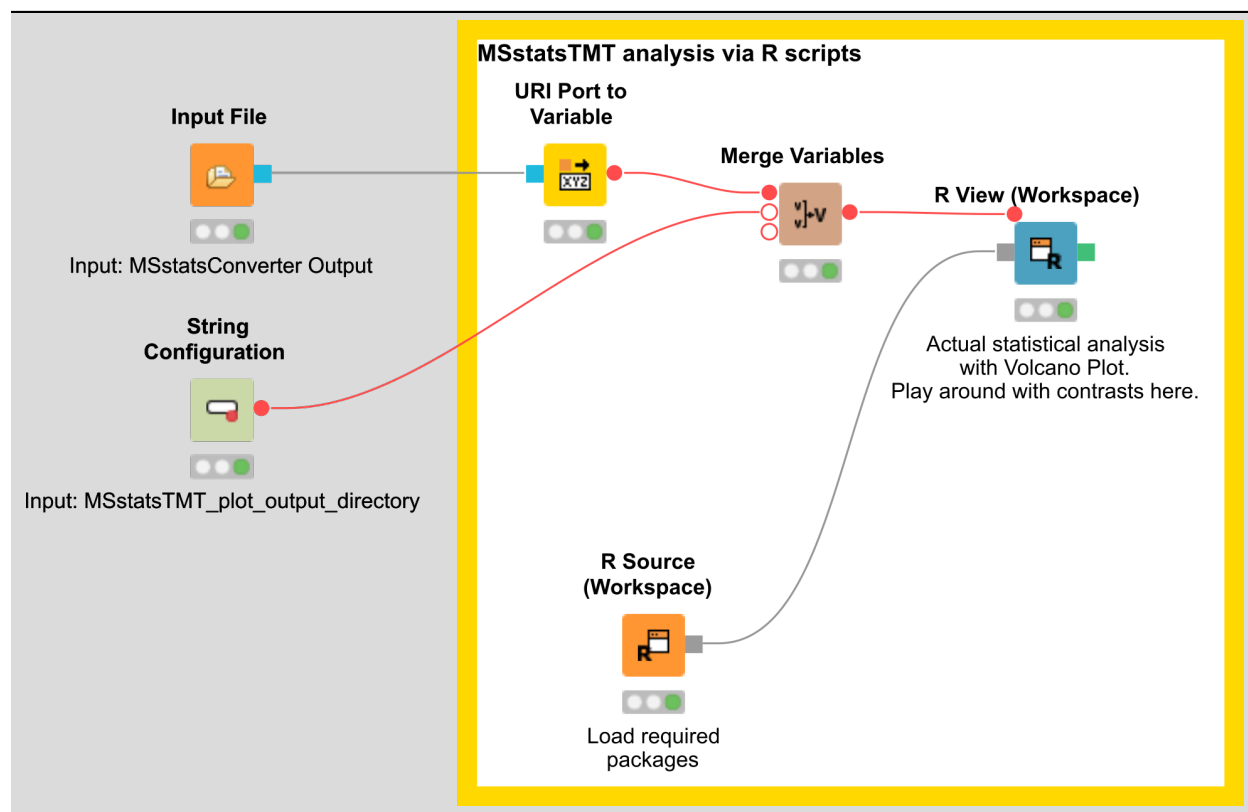
¹³ T. Huang, M. Choi, S. Hao, and O. Vitek, MSstatsTMT: Protein Significance Analysis in shotgun mass spectrometry-based proteomic experiments with tandem mass tag (TMT) labeling., (2020), doi:10.18129/B9.bioc.MSstatsTMT. 55

fraction groups. The second one 5 adds to the first by specifying specific conditions, biological replicates as well as mixtures and label for each channel. For additional information about the experimental design please see Table 3.

After running the workflow, the **MSstatsConverter** will convert the OpenMS output in addition with the experimental design to a file (.csv) which can be processed by using MSstatsTMT.

MSstatsTMT analysis

Here, we depict the analysis by MSstatsTMT using a segment of the isobaric analysis workflow (Fig. 25). The segment is available as WorkflowsMSstatsTMT.knwf.



The `OpenMStoMSstatsTMTFormat` function preprocesses the OpenMS report and converts it into the required input format for `MSstatsTMT`, by filtering based on unique peptides and measurements in each MS run.

```
processed.data <- OpenMStoMSstatsTMTFormat(data)
```

Afterwards different normalization steps are performed (global, protein, runs) as well as data imputation by using the `msstats` method. In addition peptide level data is summarized to protein level data.

```
quant.data <- proteinSummarization(processed.data,
                                   method="msstats",

                                   global_norm=TRUE,
                                   reference_norm=TRUE,

                                   MBimpute = TRUE,
                                   maxQuantileforCensored = NULL,

                                   remove_norm_channel = TRUE,
                                   remove_empty_channel = TRUE)
```

There a lot of different possibilities to configure this method please have a look at the `MSstatsTMT` package for [additional detailed information](#).

The next step is the comparisons of the different conditions, here either a pairwise comparison can be performed or a confusion matrix can be created. The goal is to detect and compare the UPS peptides spiked in at different concentrations.

```
# prepare contrast matrix
unique(quant.data$Condition)

comparison<-matrix(c(-1,0,0,1,
                     0,-1,0,1,
                     0,0,-1,1,
                     0,1,-1,0,
                     1,-1,0,0), nrow=5, byrow = T)

# Set the names of each row
row.names(comparison)<- contrasts <- c("1-0125",
                                       "1-05",
                                       "1-0667",
                                       "05-0667",
                                       "0125-05")

# Set the column names
colnames(comparison)<- c("0.125", "0.5", "0.667", "1")
```

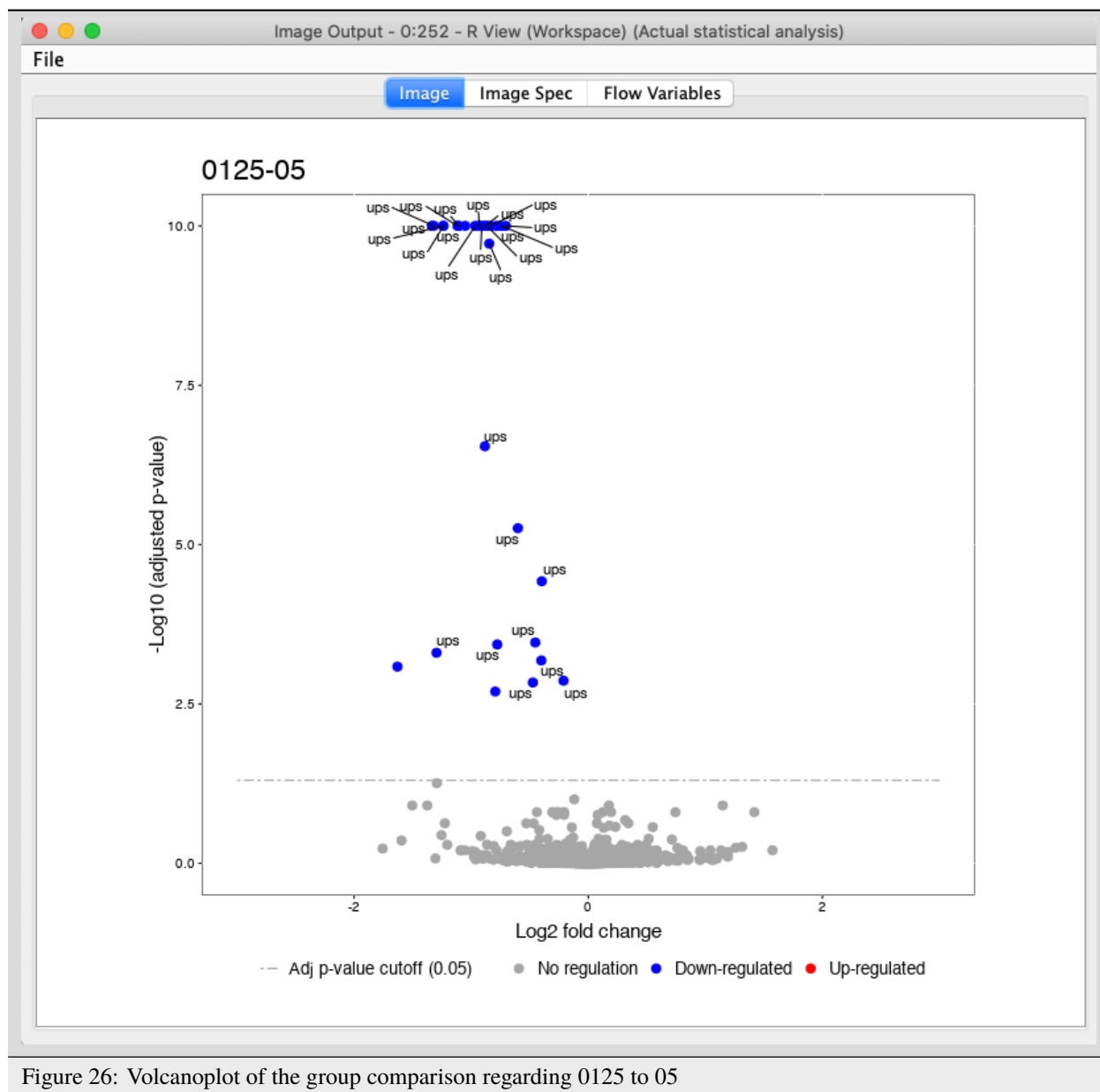
The constructed confusion matrix is used in the `groupComparisonTMT` function to test for significant changes in protein abundance across conditions based on a family of linear mixed-effects models in TMT experiments.

```
data.res <- groupComparisonTMT(data = quant.data,  
                               contrast.matrix = comparison,  
  
                               moderated = TRUE, # do moderated t test  
  
                               adj.method = "BH") # multiple comparison adjustment  
data.res <- data.res %>% filter(!is.na(Protein))
```

In the next step the comparison can be plotted using the `groupComparisonPlots` function by `MSstats`.

```
library(MSstats)  
groupComparisonPlots(data=data.res.mod, type="VolcanoPlot", address=F, which.Comparison_  
↪ = "0125-05", sig = 0.05)
```

Here, we have an example output of the **R View**, which depicts the significant regulated UPS proteins in the comparison of 125 to 05 (Fig. 26).



All plots are saved to the in the beginning specified output directory in addition.

Note

The isobaric analysis does not always have to be performed on protein level, for example for phosphoproteomics studies one is usually interested on the peptide level - in addition inference on peptides with post-translational modification is not straight forward. Here, we present an additional workflow on peptide level, which can potentially be adapted and used for such cases. Please see [WorkflowsIdentificationquantificationisobaricMSstatsTMT](#)

Label-free quantification of metabolites

Introduction

Quantification and identification of chemical compounds are basic tasks in metabolomic studies. In this tutorial session we construct a UPLC-MS based, label-free quantification and identification workflow. Following quantification and identification we then perform statistical downstream analysis to detect quantification 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. The goal of this tutorial is to detect and quantify these differential spike-in compounds against the complex background.

Basics of non-targeted metabolomics data analysis

For the metabolite quantification we choose an approach similar to the one used for peptides, but this time based on the OpenMS **FeatureFinderMetabo** method. This feature finder again collects peak picked data into individual mass traces. The reason why we need a different feature finder for metabolites lies in the step after trace detection: the aggregation of isotopic traces belonging to the same compound ion into the same feature. Compared to peptides with their average model, small molecules have very different isotopic distributions. To group small molecule mass traces correctly, an aggregation model tailored to small molecules is thus needed.

- Create a new workflow called for instance "Metabolomics".
- Add an **Input File** node and configure it with one mzML file from the `Example_DataMetabolomicsdatasets`.
- Add a **FeatureFinderMetabo** node (from **Community Nodes > OpenMS > Quantitation**) and connect the first output port of the **Input File** to the **FeatureFinderMetabo**.
- For an optimal result adjust the following settings. Please note that some of these are advanced parameters.
- Connect a **Output Folder** to the output of the **FeatureFinderMetabo** (see Fig. 27).

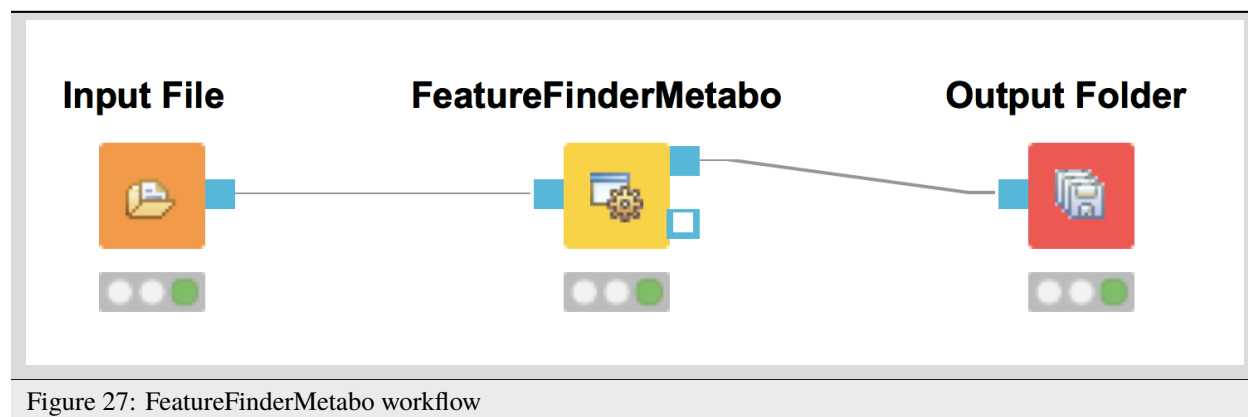


Figure 27: FeatureFinderMetabo workflow

In the following advanced parameters will be highlighted. These parameter can be altered if the `Show advanced parameter` field in the specific tool is activated.

parameter	value
<i>algorithm→common→chrom_fwhm</i>	8.0
<i>algorithm→mtd→trace_termination_criterion</i>	sample_rate
<i>algorithm→mtd→min_trace_length</i>	3.0
<i>algorithm→mtd→max_trace_length</i>	600.0
<i>algorithm→epd→width_filtering</i>	off
<i>algorithm→ffm→report_convex_hulls</i>	true

The parameters change the behavior of **FeatureFinderMetabo** as follows:

- **chrom_fwhm**: The expected chromatographic peak width in seconds.
- **trace_termination_criterion**: In the first stage **FeatureFinderMetabo** assembles mass traces with a pre-defined mass accuracy. If this parameter is set to 'outlier', the extension of a mass trace is stopped after a predefined number of consecutive outliers is found. If this parameter is set to 'sample_rate', the extension of a mass trace is stopped once the ratio of collected peaks versus visited spectra falls below the ratio given by `min_sample_rate`.
- **min_trace_length**: Minimal length of a mass trace in seconds. Choose a small value, if you want to identify low-intensity compounds.
- **max_trace_length**: Maximal length of a mass trace in seconds. Set this parameter to -1 to disable the filtering by maximal length.
- **width_filtering**: **FeatureFinderMetabo** can remove features with unlikely peak widths from the results. If activated it will use the interval provided by the parameters `min_fwhm` and `max_fwhm`.
- **report_convex_hulls**: If set to true, convex hulls including mass traces will be reported for all identified features. This increases the output size considerably.

The output file `.featureXML` can be visualized with TOPPView on top of the used `.mzML` file - in a so called layer - to look at the identified features.

First start TOPPView and open the example `.mzML` file (see Fig. 28). Afterwards open the `.featureXML` output as new layer (see Fig. 29). The overlay is depicted in Figure 30. The zoom of the `.mzML` - `.featureXML` overlay shows the individual mass traces and the assembly of those in a feature (see Fig. 31).

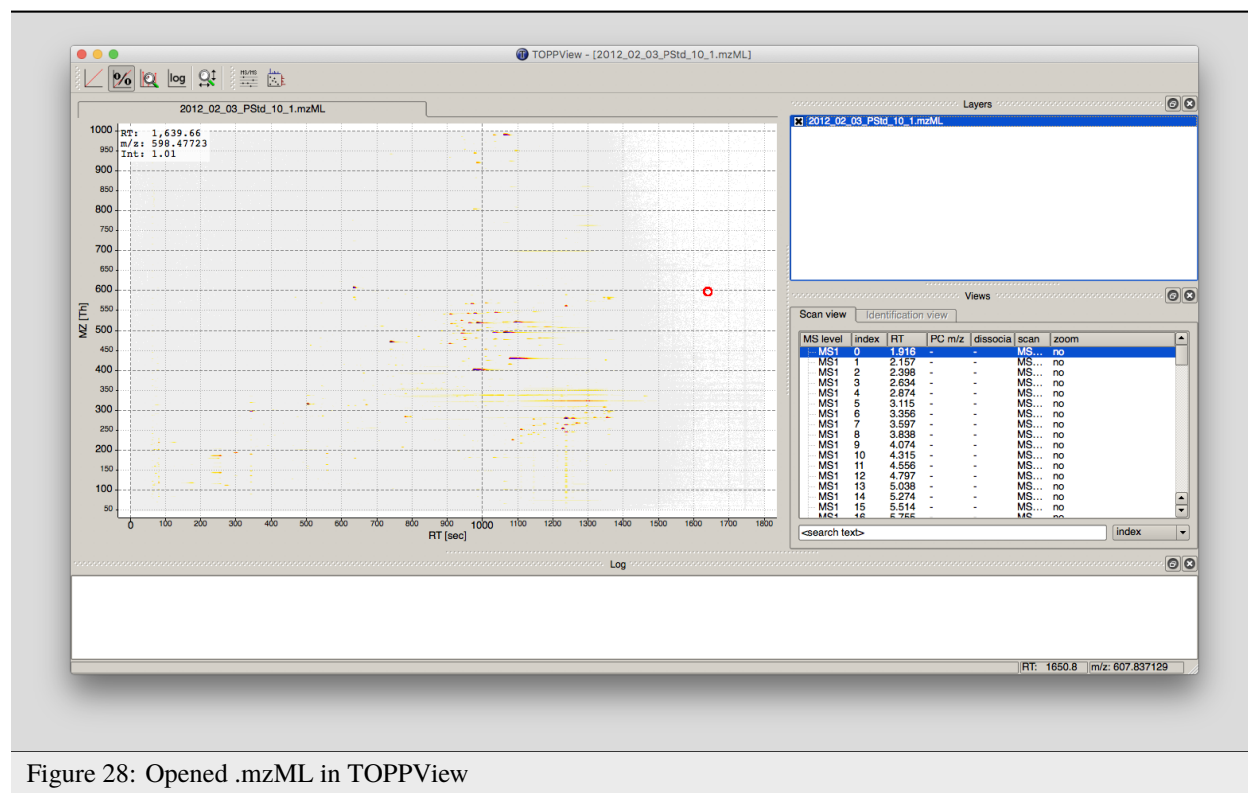


Figure 28: Opened .mzML in TOPPView

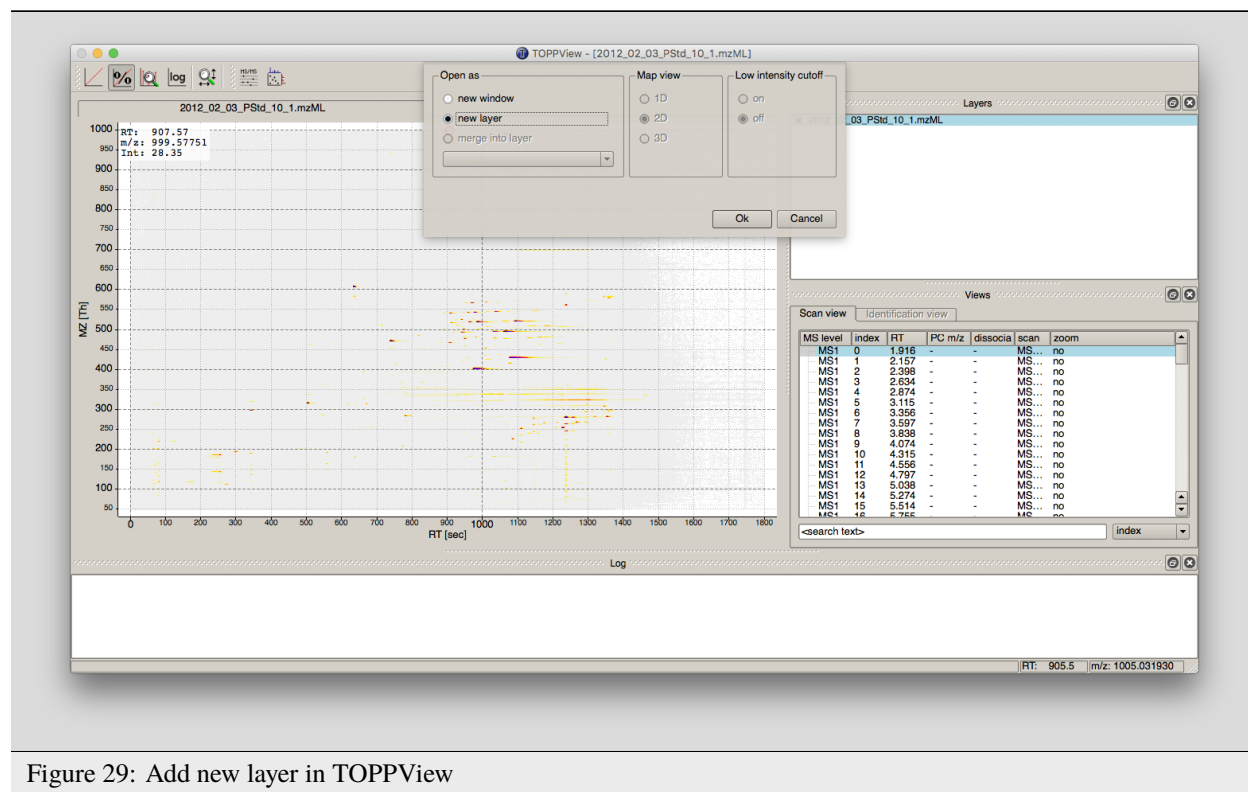


Figure 29: Add new layer in TOPPView

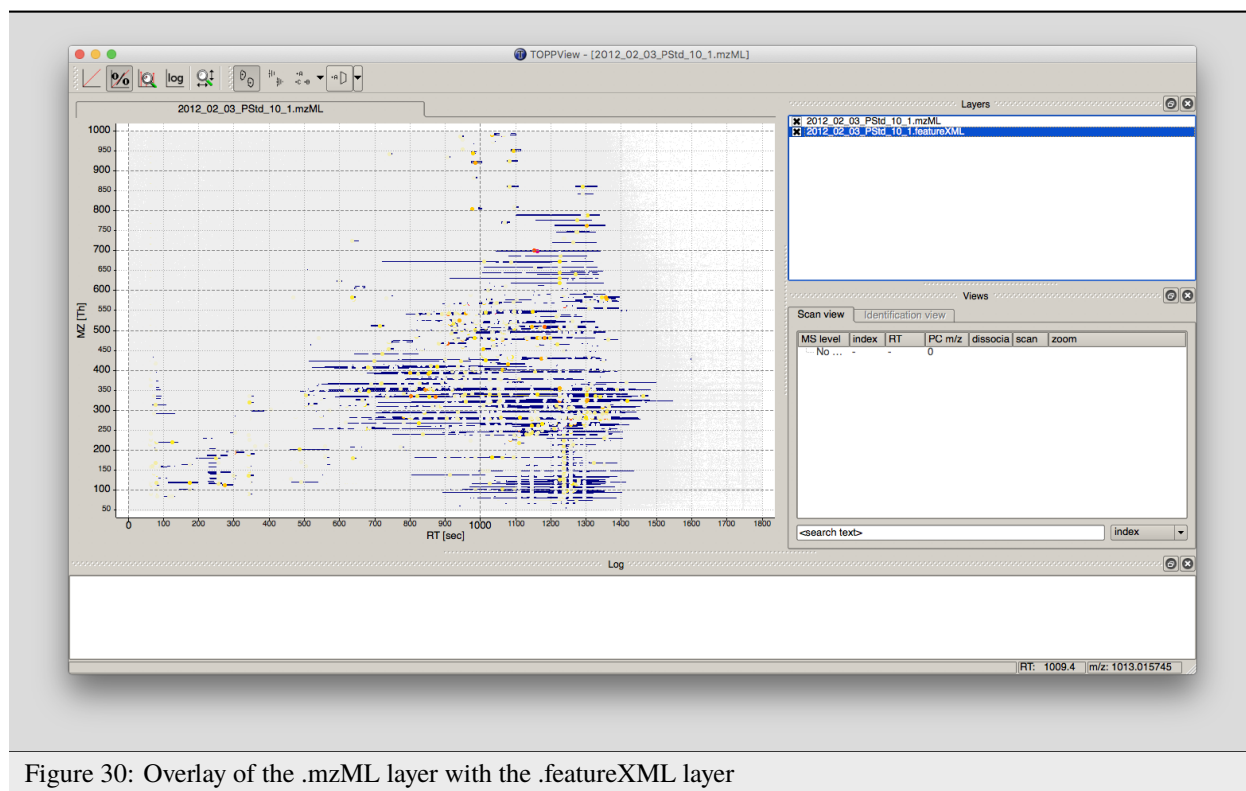


Figure 30: Overlay of the .mzML layer with the .featureXML layer

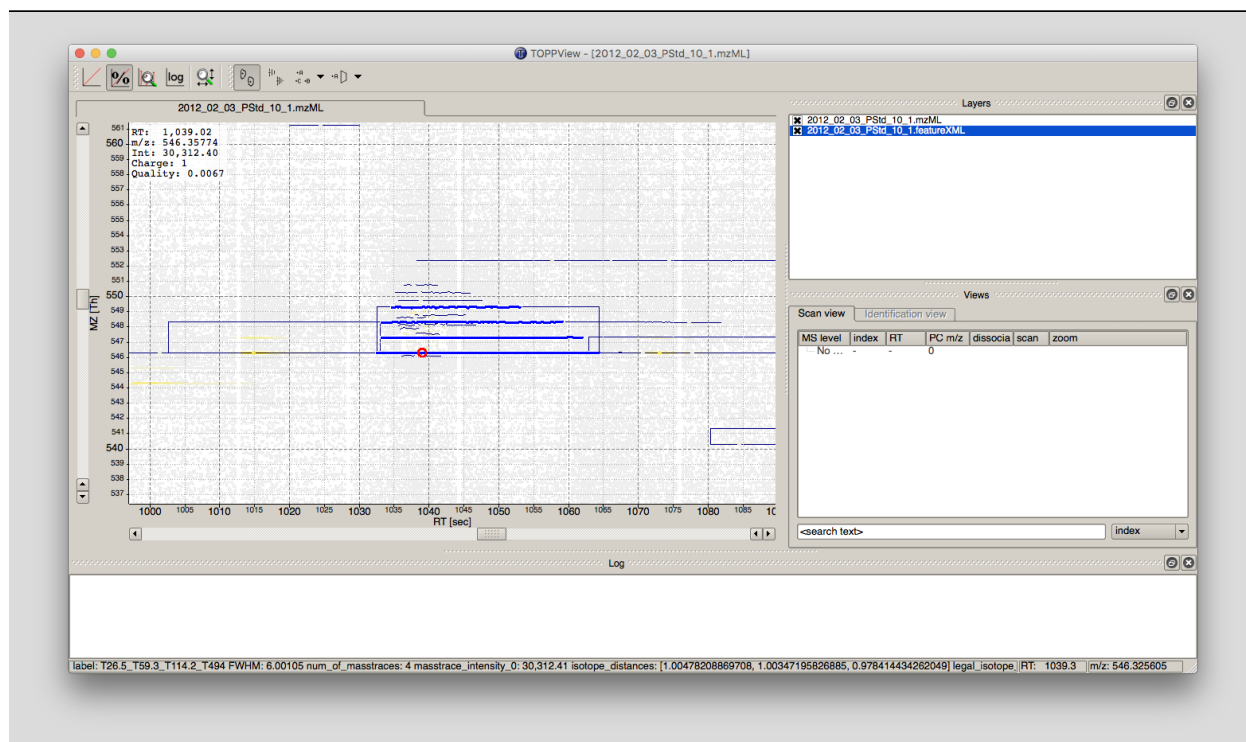


Figure 31: Zoom of the overlay of the .mzML with the .featureXML layer. Here the individual isotope traces (blue lines) are assembled into a feature here shown as convex hull (rectangular box).

The workflow can be extended for multi-file analysis, here an **Input Files** node is to be used instead of the **Input File** node. In front of the **FeatureFinderMetabo**, a **ZipLoopStart** and behind **ZipLoopEnd** has to be used, since **FeatureFinderMetabo** will analysis on file to file bases.

To facilitate the collection of features corresponding to the same compound ion across different samples, an alignment of the samples' feature maps along retention time is often helpful. In addition to local, small-scale elution differences, one can often see constant retention time shifts across large sections between samples. We can use linear transformations to correct for these large scale retention differences. This brings the majority of corresponding compound ions close to each other. Finding the correct corresponding ions is then faster and easier, as we don't have to search as far around individual features.

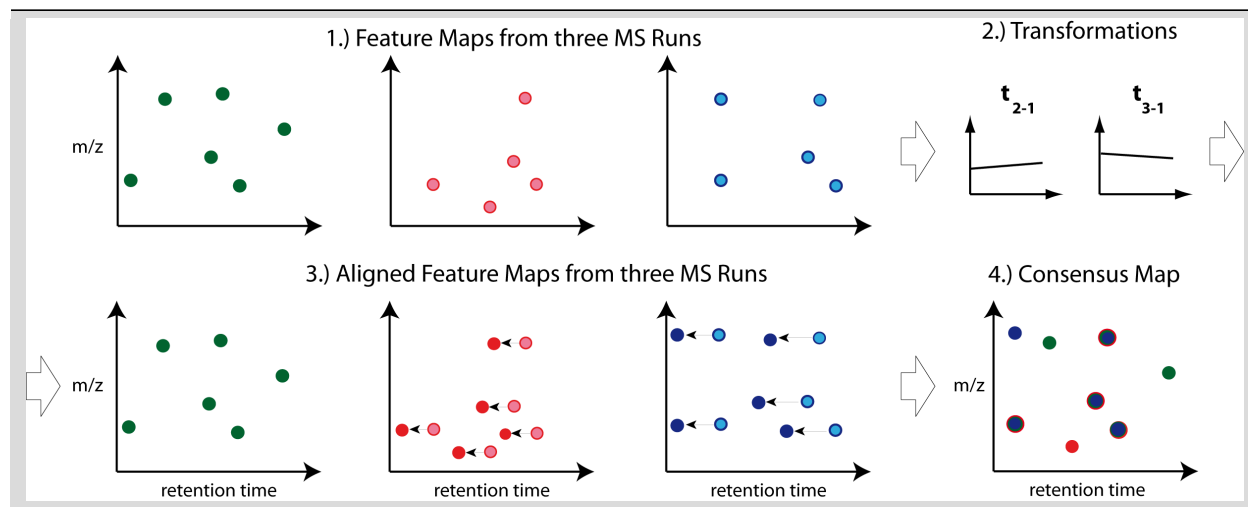


Figure 32: The first feature map is used as a reference to which other maps are aligned. The calculated transformation brings corresponding features into close retention time proximity. Linking of these features form a so-called consensus features of a consensus map.

- 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
<code>algorithm → max_num_peaks_considered</code>	1
<code>algorithm → superimposer → mz_pair_max_distance</code>	0.005
<code>algorithm → superimposer → num_used_points</code>	10000
<code>algorithm → pairfinder → distance_RT → max_difference</code>	20.0
<code>algorithm → pairfinder → distance_MZ → max_difference</code>	20.0
<code>algorithm → pairfinder → distance_MZ → unit</code>	ppm

MapAlignerPoseClustering provides an algorithm to align the retention time scales of multiple input files, correcting shifts and distortions between them. Retention time adjustment may be necessary to correct for chromatography differences e.g. before data from multiple LC-MS runs can be combined (feature linking). The alignment algorithm implemented here is the pose clustering algorithm.

The parameters change the behavior of **MapAlignerPoseClustering** as follows:

- **max_num_peaks_considered**: The maximal number of peaks/features to be considered per map. To use all, set this parameter to -1.
- **mz_pair_max_distance**: Maximum of m/z deviation of corresponding elements in different maps. This condition applies to the pairs considered in hashing.

- **num_used_points**: Maximum number of elements considered in each map (selected by intensity). Use a smaller number to reduce the running time and to disregard weak signals during alignment.
- **distance_RT** → **max_difference**: Features that have a larger RT difference will never be paired.
- **distance_MZ** → **max_difference**: Features that have a larger m/z difference will never be paired.
- **distance_MZ** → **unit**: Unit used for the parameter distance_MZ max_difference, either Da or ppm.

The next step after retention time correction is the grouping of corresponding features in multiple samples. In contrast to the previous alignment, we assume no linear relations of features across samples. The used method is tolerant against local swaps in elution order.

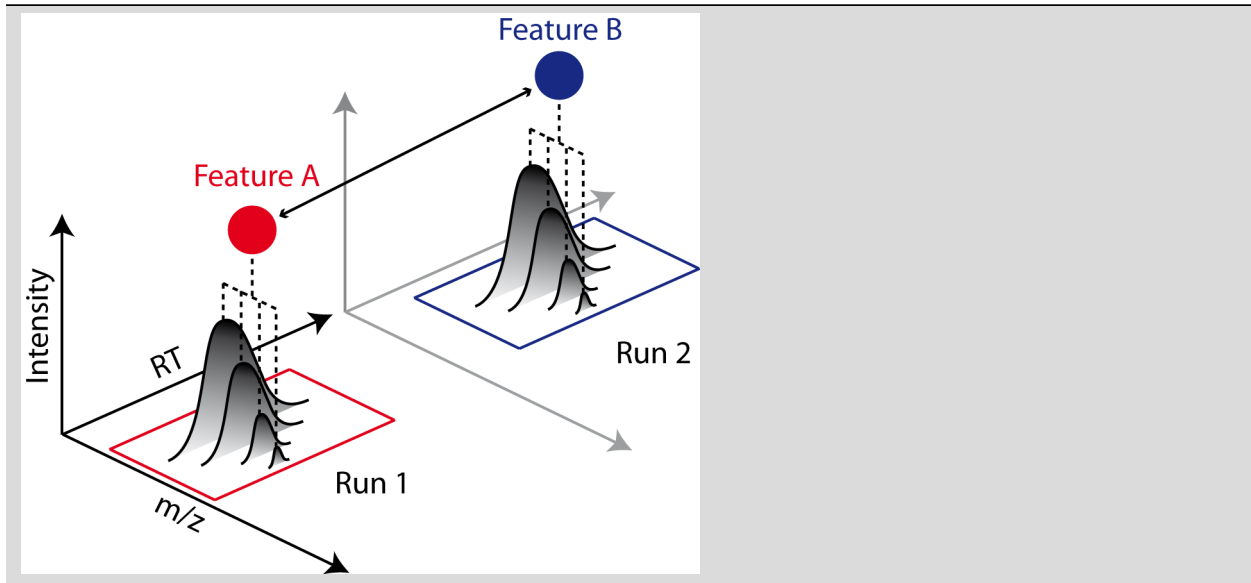


Figure 33: Features A and B correspond to the same analyte. The linking of features between runs (indicated by an arrow) allows comparing feature intensities.

- After the **MapAlignerPoseClustering** node, add a **FeatureLinkerUnlabeledQT** node (**Community Nodes > OpenMS>Map Alignment**) and adjust the following settings:

parameter	value
<i>algorithm</i> → <i>distance_RT</i> → <i>max_difference</i>	40
<i>algorithm</i> → <i>distance_MZ</i> → <i>max_difference</i>	20
<i>algorithm</i> → <i>distance_MZ</i> → <i>unit</i>	ppm

The parameters change the behavior of **FeatureLinkerUnlabeledQT** as follows (similar to the parameters we adjusted for **MapAlignerPoseClustering**):

- **distance_RT** → **max_difference**: Features that have a larger RT difference will never be paired.
- **distance_MZ** → **max_difference**: Features that have a larger m/z difference will never be paired.
- **distance_MZ** → **unit**: Unit used for the parameter distance_MZ max_difference, either Da or ppm.
- After the **FeatureLinkerUnlabeledQT** node, 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.

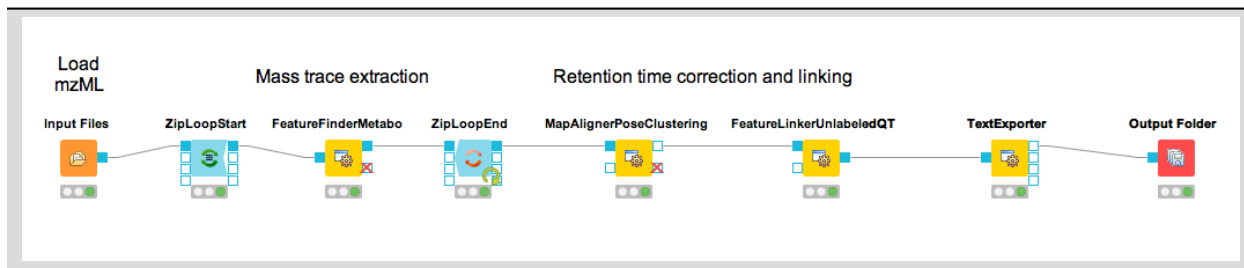


Figure 34: Label-free quantification workflow for metabolites.

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 34. In the following section we will try to identify those metabolites.

The **FeatureLinkerUnlabeledQT** output can be visualized in TOPPView on top of the input and output of the **FeatureFinderMetabo** (see Fig 35).

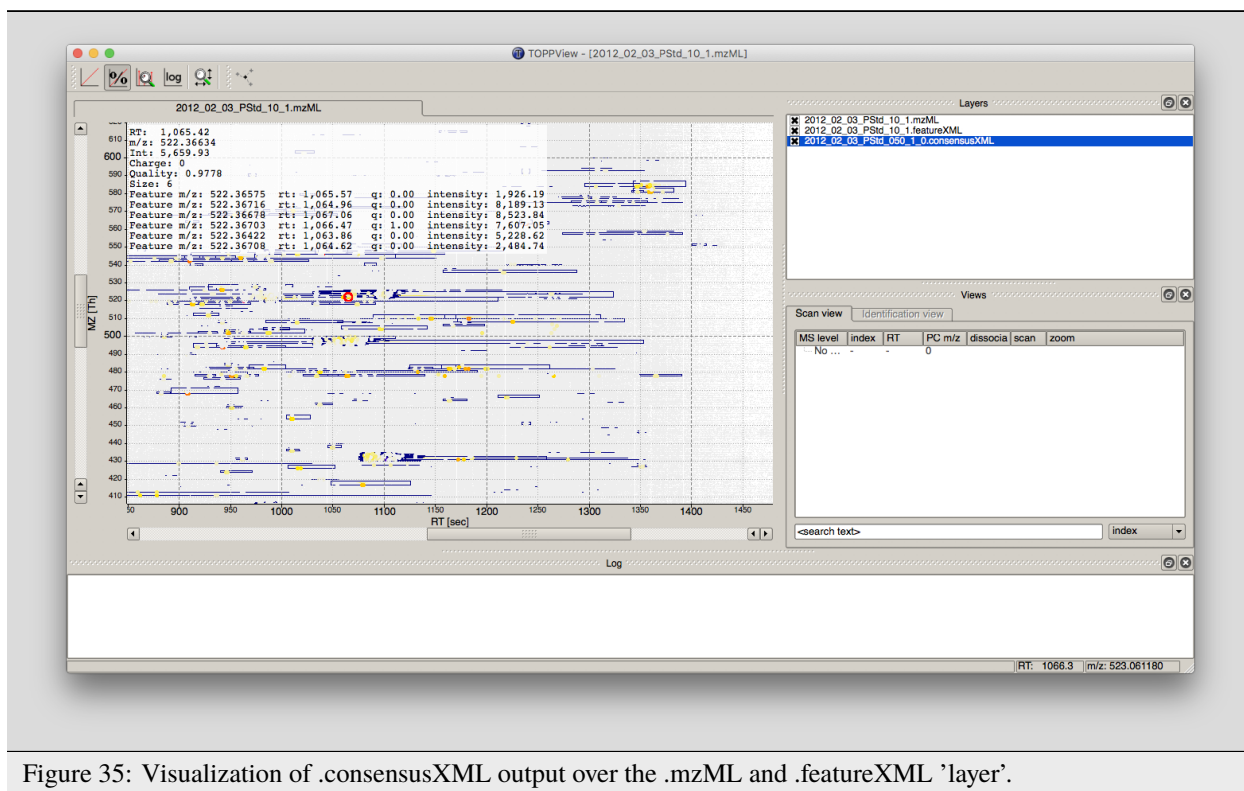


Figure 35: Visualization of .consensusXML output over the .mzML and .featureXML 'layer'.

Basic metabolite identification

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 multiple tools, including search by mass: the **AccurateMassSearch** node searches observed masses against the Human Metabolome Database (HMDB)^{14, 15, 16}. We start with the workflow from the previous section (see Figure 34).

- Add a **FileConverter** node (**Community Nodes > OpenMS > File Handling**) and connect the output of the **FeatureLinkerUnlabeledQT** to the incoming port.
- Open the Configure dialog of the **FileConverter** node and select the tab **OutputTypes**. In the drop down list for **FileConverter.l.out** select **featureXML**.
- Add an **AccurateMassSearch** node (**Community Nodes > OpenMS > Utilities**) and connect the output of the **FileConverter** node to the first port of the **AccurateMassSearch** node.
- Add four **Input File** nodes and configure them with the following files:
 - **Example_DataMetabolomicsdatabasesPositiveAdducts.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.
 - **Example_DataMetabolomicsdatabasesNegativeAdducts.tsv** This file specifies the list of adducts that are considered in the negative mode analogous to the positive mode.
 - **Example_DataMetabolomicsdatabasesHMDBMappingFile.tsv** This file contains information from a metabolite database in this case from HMDB. It has three (or more) tab-separated columns: mass, formula, and identifier(s). This allows for an efficient search by mass.
 - **Example_DataMetabolomicsdatabasesHMDB2StructMapping.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** node.

The result of the **AccurateMassSearch** node is in the **mzTab** format¹⁷ 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 36.

¹⁴ 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. 69

¹⁵ 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. 69

¹⁶ 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. 69

¹⁷ 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. 69

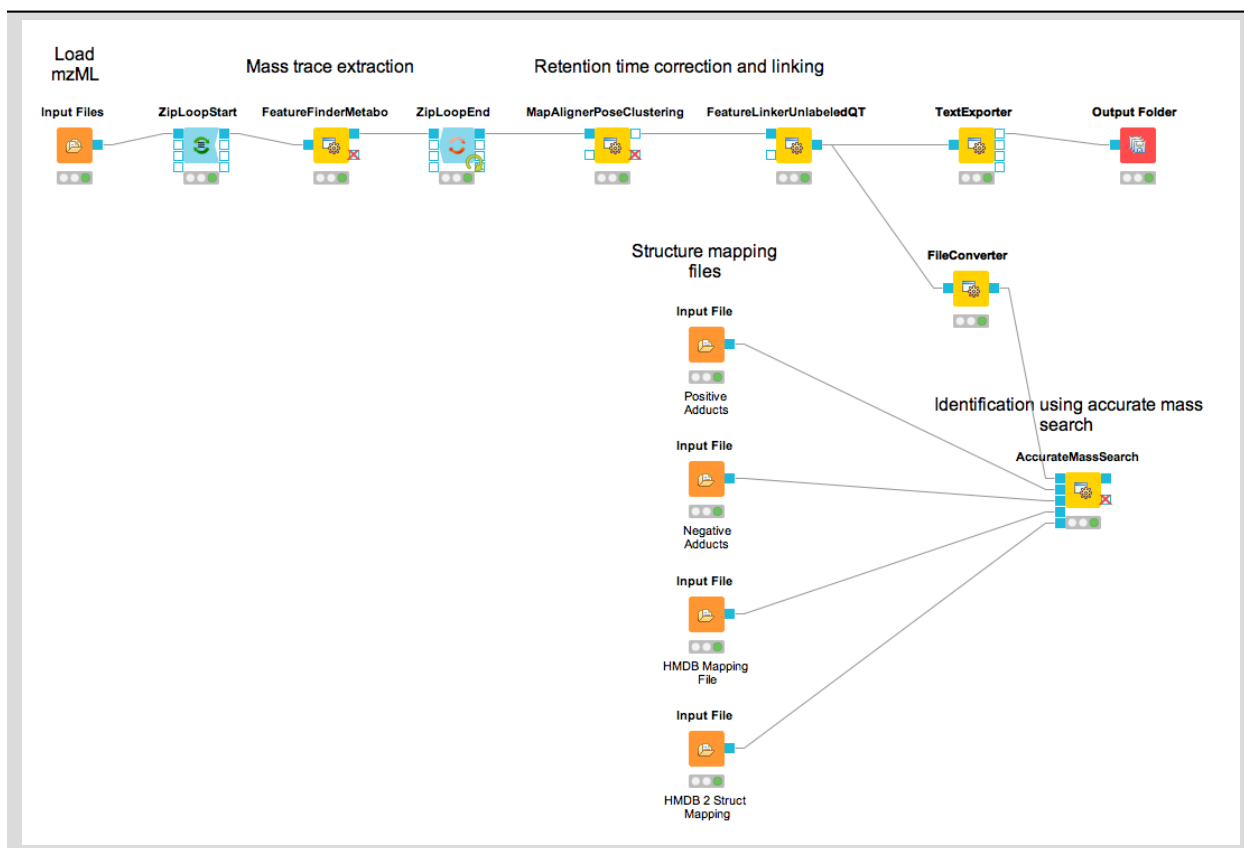


Figure 36: Label-free quantification and identification workflow for metabolites.

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 process only KNIME tables. To convert these files into KNIME tables we need two different nodes. For the **AccurateMassSearch** results, we use the **MzTabReader** node (**Community Nodes > OpenMS > Conversion > mzTab**) and its **Small Molecule Section** port. 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. The retention time values are exported as a list using the **MzTabReader** based on the current PSI-Standard. This has to be parsed using the **SplitCollectionColumn**, which outputs a "Split Value 1" based on the first entry in the retention time list, which has to be renamed to retention time using the **ColumnRename**. You can now combine both tables using the **Joiner** node (**Manipulation > Column > Split & Combine**) and configure it to match the m/z and retention time values of the respective tables. The full workflow is shown in Figure 37.

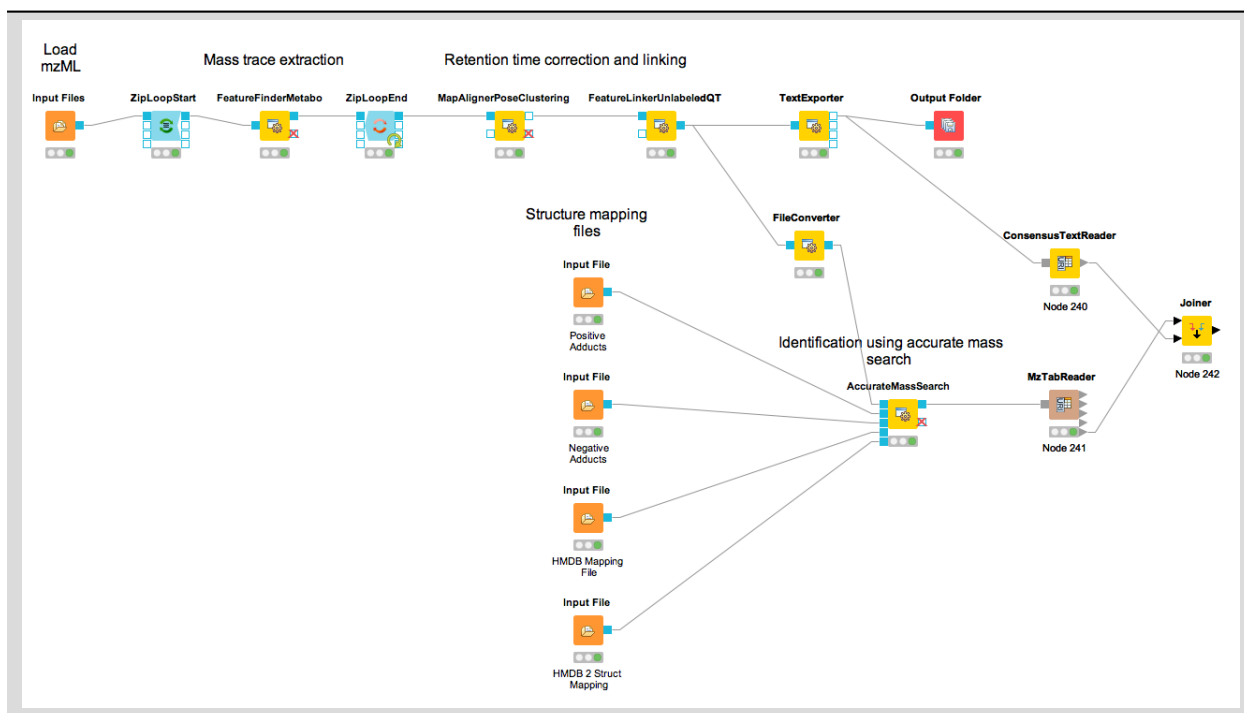


Figure 37: Label-free quantification and identification workflow for metabolites that loads the results into KNIME and joins the tables.

Adduct grouping

Metabolites commonly co-elute as ions with different adducts (e.g., glutathione+H, glutathione+Na) or with charge-neutral modifications (e.g., water loss). Grouping such related ions allows to leverage information across features. For example, a low intensity, single trace feature could still be assigned a charge and adduct due to a matching high-quality feature. Several OpenMS tools, such as **AccurateMassSearch**, can use this information to, for example, narrow down candidates for identification.

For this grouping task, we provide the **MetaboliteAdductDecharger** node. Its method explores the combinatorial space of all adduct combinations in a charge range for optimal explanations. Using defined adduct probabilities, it assigns co-eluting features having suitable mass shifts and charges those adduct combinations which maximize overall ion probabilities.

The tool works natively with featureXML data, allowing the use of reported convex hulls. On such a single-sample level, co-elution settings can be chosen more stringently, as ionization-based adducts should not influence the elution time: Instead, elution differences of related ions should be due to slightly differently estimated times for their feature centroids.

Alternatively, consensusXML data from feature linking can be converted for use, though with less chromatographic information. Here, the elution time averaging for features linked across samples, motivates wider co-elution tolerances.

The two main tool outputs are a consensusXML file with compound groups of related input ions, and a featureXML containing the input file but annotated with inferred adduct information and charges.

Options to respect or replace ion charges or adducts allow for example:

- Heuristic but faster, iterative adduct grouping (**MetaboliteAdductDecharger** → **MetaboliteFeatureDeconvolution** → **q_try** set to “feature”) by chaining multiple **MetaboliteAdductDecharger** nodes with growing adduct sets, charge ranges or otherwise relaxed tolerances.

- More specific feature linking (**FeatureLinkerUnlabeledQT** → **algorithm** → **ignore_adduct** set to “false”)

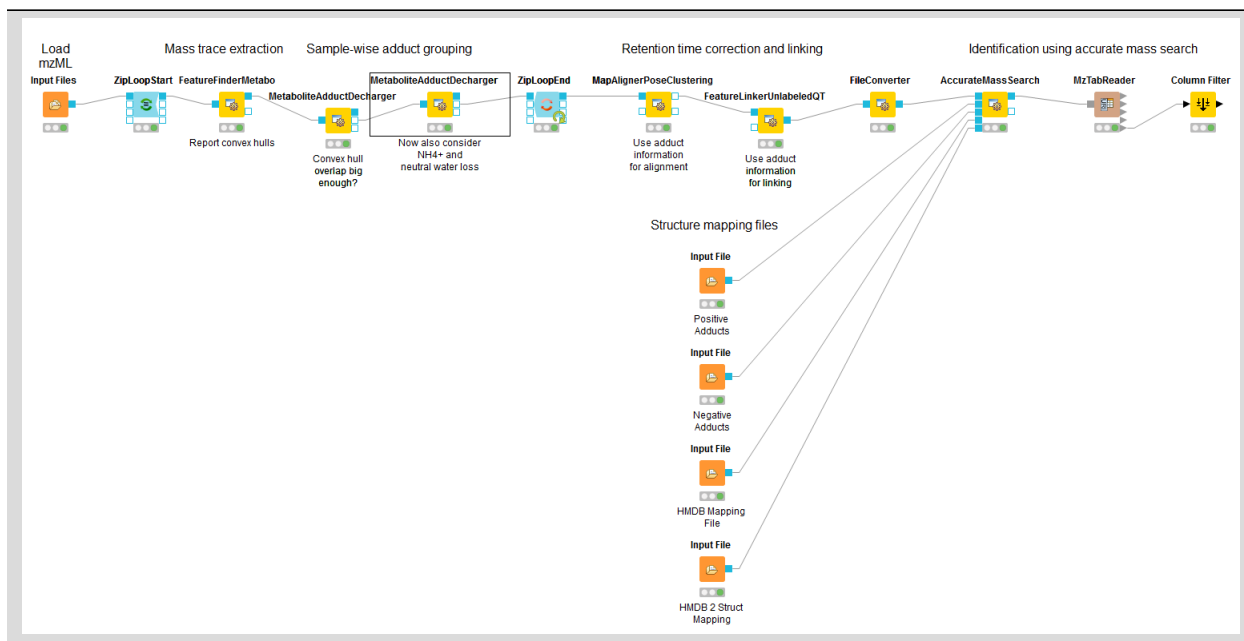


Figure 38: Metabolite Adduct Decharger adduct grouping workflow.

Task

A modified metabolomics workflow with exemplary MetaboliteAdductDecharger use and parameters is provided in WorkflowsMetaboliteAdductGrouping.knwf. Run the workflow, inspect tool outputs and compare **AccurateMassSearch** results with and without adduct grouping.

Visualizing 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 (**Chemistry > Translators**) together with subsequent cheminformatics nodes (e.g. **RDKit From Molecule**(**Community Nodes > RDKit > Converters**)) 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.

Task

Try to compute and visualize the m/z and retention time error of the different feature elements (from the input maps) of each consensus feature. Hint: A nicely configured **Math Formula (Multi Column)** node should suffice.

Spectral library search

Identifying metabolites using only the accurate mass may lead to ambiguous results. In practice, additional information (e.g. the retention time) is used to further narrow down potential candidates. Apart from MS1-based features, tandem mass spectra (MS2) of metabolites provide additional information. In this part of the tutorial, we take a look on how metabolite spectra can be identified using a library of previously identified spectra.

Because these libraries tend to be large we don't distribute them with OpenMS.

Task

Construct the workflow as shown in Fig. 39. Use the file `ExampleData\Metabolomics\datasets\MetaboliteIDSpectraDB-positive.mzML` as input for your workflow. You can use the spectral library from `ExampleData\Metabolomics\database\MetaboliteSpectralDB.mzML` as second input. The first input file contains tandem spectra that are identified by the **MetaboliteSpectralMatcher**. The resulting mzTab file is read back into a KNIME table. The retention time values are exported as a list based on the current PSI-Standard. This has to be parsed using the **SplitCollectionColumn**, which outputs a "Split Value 1" based on the first entry in the retention time list, which has to be renamed to retention time using the **ColumnRename** before it is stored in an Excel table. Make sure that you connect the **MzTabReader** port corresponding to the Small Molecule Section to the **Excel writer (XLS)**. Please select the "add column headers" option in the **Excel writer (XLS)**.

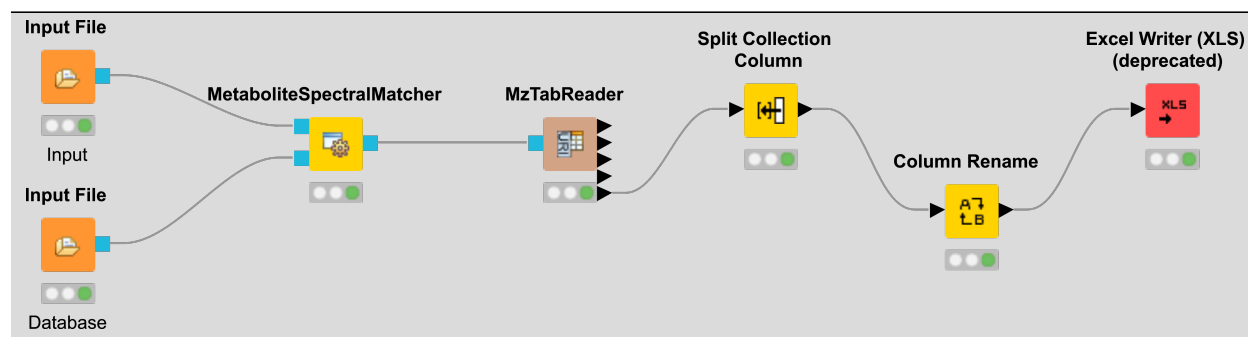


Figure 39: Spectral library identification workflow.

Run the workflow and inspect the output.

Manual validation

In metabolomics, matches between tandem spectra and spectral libraries are manually validated. Several commercial and free online resources exist which help in that task. Some examples are:

- mzCloud contains only spectra from Thermo Orbitrap instruments. The webpage requires Microsoft Silverlight which currently does not work in modern browsers (see the following [link](#)).
- MassBank North America (MoNA) has spectra from different instruments but falls short in number of spectra (compared to Metlin and mzCloud). See the following [link](#).
- METLIN includes 961,829 molecules ranging from lipids, steroids, metabolites, small peptides, carbohydrates, exogenous drugs and toxicants. In total over 14,000 metabolites.

Here, we will use METLIN to manually validate metabolites.

Task

Check in the .xlsx output from the Excel writer (XLS) if you can find glutathione. Use the retention time column to find the spectrum in the mzML file. Here open the file in the Example_DataMetabolomicsdatasetsMetaboliteIDSpectraDBpositive.mzML in TOPPView. The MSMS spectrum with the retention time of 67.6 s is used as example. The spectrum can be selected based on the retention time in the scan view window. Therefore the MS1 spectrum with the retention time of 66.9 s has to be double clicked and the MSMS spectra recorded in this time frame will show up. Select the tandem spectrum of Glutathione, but do not close TOPPView, yet.

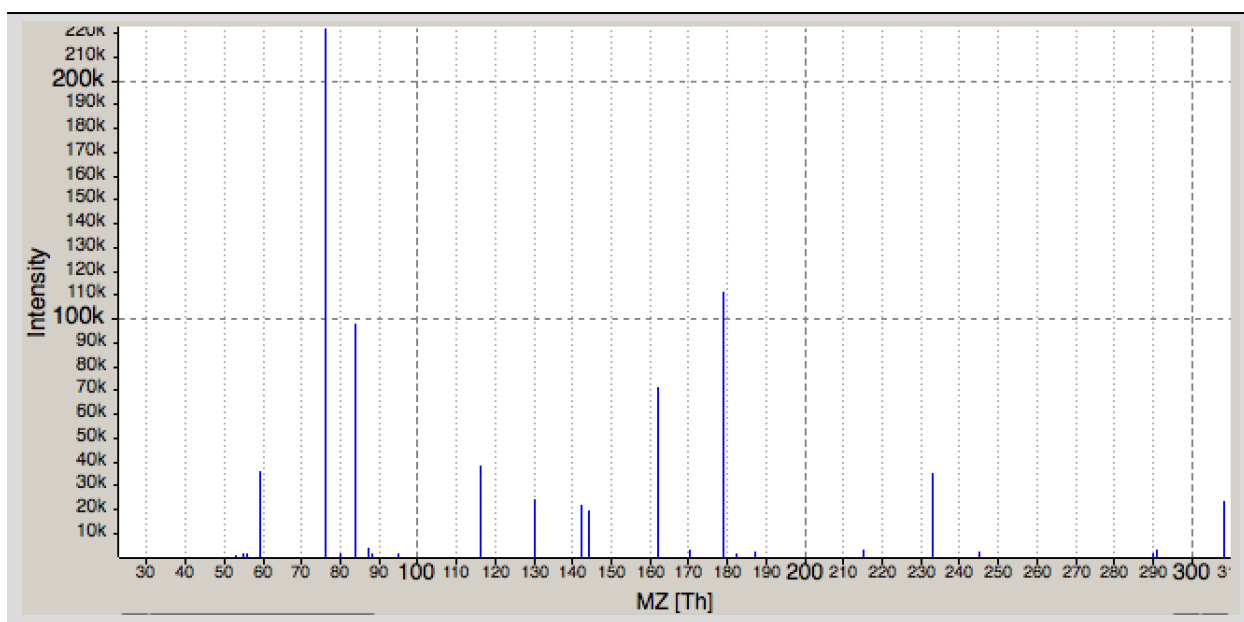


Figure 40: Tandem spectrum of glutathione. Visualized in TOPPView.

Task

On the METLIN homepage search for **Name** Glutathione using the **Advanced Search**. See the [link](#). Note that free registration is required. Which collision energy (and polarity) gives the best (visual) match to your experimental spectrum in TOPPView? Here you can compare the fragmentation patterns in both spectra shown by the Intensity or relative Intensity, the m/z of a peak and the distance between peaks. Each distance between two peaks corresponds to a fragment of elemental composition (e.g., NH₂ with the charge of one would have mass of two peaks of 16.023 Th).

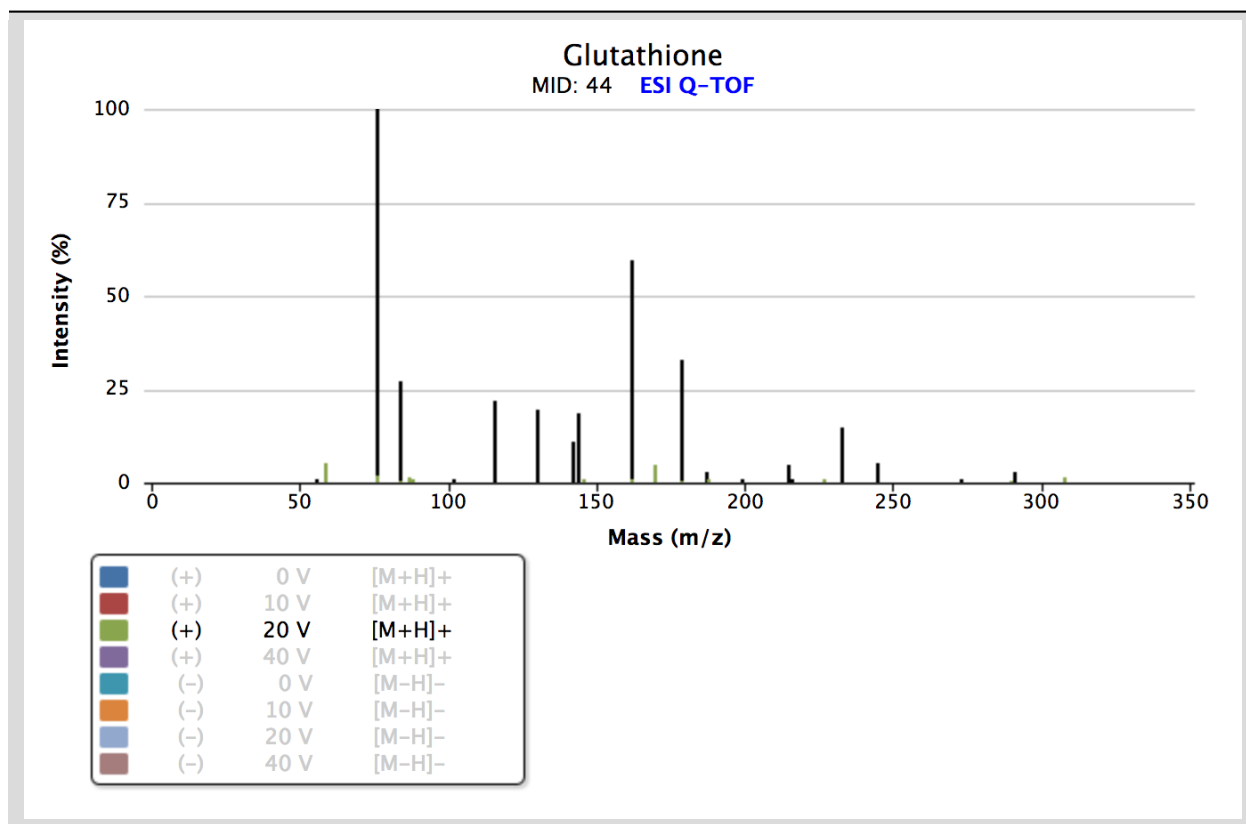


Figure 41: Tandem spectrum of glutathione. Visualized in Metlin. Note that several fragment spectra from varying collision energies are available.

De novo identification

Another method for MS2 spectra-based metabolite identification is de novo identification. This approach can be used in addition to the other methods (accurate mass search, spectral library search) or individually if no spectral library is available. In this part of the tutorial, we discuss how metabolite spectra can be identified using de novo tools. To this end, the tools SIRIUS and CSI:FingerID^(18,19,20) were integrated in the OpenMS Framework as **SiriusAdapter**. SIRIUS uses isotope pattern analysis to detect the molecular formula and further analyses the fragmentation pattern of a compound using fragmentation trees. CSI:FingerID is a method for searching a fingerprint of a small molecule (metabolite) in a molecular structure database. The node **SiriusAdapter** is able to work in different modes depending on the provided input.

- Input: mzML - **SiriusAdapter** will search all MS2 spectra in a map.
- Input: mzML, featureXML (FeatureFinderMetabo) - **SiriusAdapter** can use the provided feature information to reduce the search space to valid features with MS2 spectra. Additionally it can use the isotopic trace information.
- Input: mzML, featureXML (FeatureFinderMetabo / MetaboliteAdductDecharger / AccurateMassSearch) - **SiriusAdapter** can use the feature information as mentioned above together with feature adduct information from adduct grouping or previous identification.

¹⁸ S. Böcker, M. C. Letzel, Z. Lipták, and A. Pervukhin, SIRIUS: Decomposing isotope patterns for metabolite identification, *Bioinformatics* 25(2), 218–224 (2009), doi:10.1093/bioinformatics/btn603. 75

¹⁹ S. Böcker and K. Dührkop, Fragmentation trees reloaded, *J. Cheminform.* 8(1), 1–26 (2016), doi:10.1186/s13321-016-0116-8. 75

²⁰ K. Dührkop, H. Shen, M. Meusel, J. Rousu, and S. Böcker, Searching molecular structure databases with tandem mass spectra using CSI:FingerID, *Proc. Natl. Acad. Sci.* 112(41), 12580–12585 (oct 2015), doi:10.1073/pnas.1509788112. 75

By using a mzML and featureXML, SIRIUS gains a lot of additional information by using the OpenMS tools for preprocessing.

Task

Construct the workflow as shown in Fig. 42. Example_DataMetabolomicsdatasets Use the file `MetaboliteDeNovoID.mzML` as input for your workflow.

Below we show an example workflow for de novo identification (Fig. 42). Here, the node **FeatureFinderMetabo** is used for feature detection to annotate analytes in mz, rt, intensity and charge. This is followed by adduct grouping, trying to assess possible adducts based on the feature space using the **MetaboliteAdductDecharger**. In addition, the **HighResPrecursorMassCorrector** can use the newly generated feature information to map MS2 spectra, which were measured on one of the isotope traces to the monoisotopic precursor. This helps with feature mapping and analyte identification in the **SiriusAdapter** due to the usage of additional MS2 spectra that belong to a specific feature.

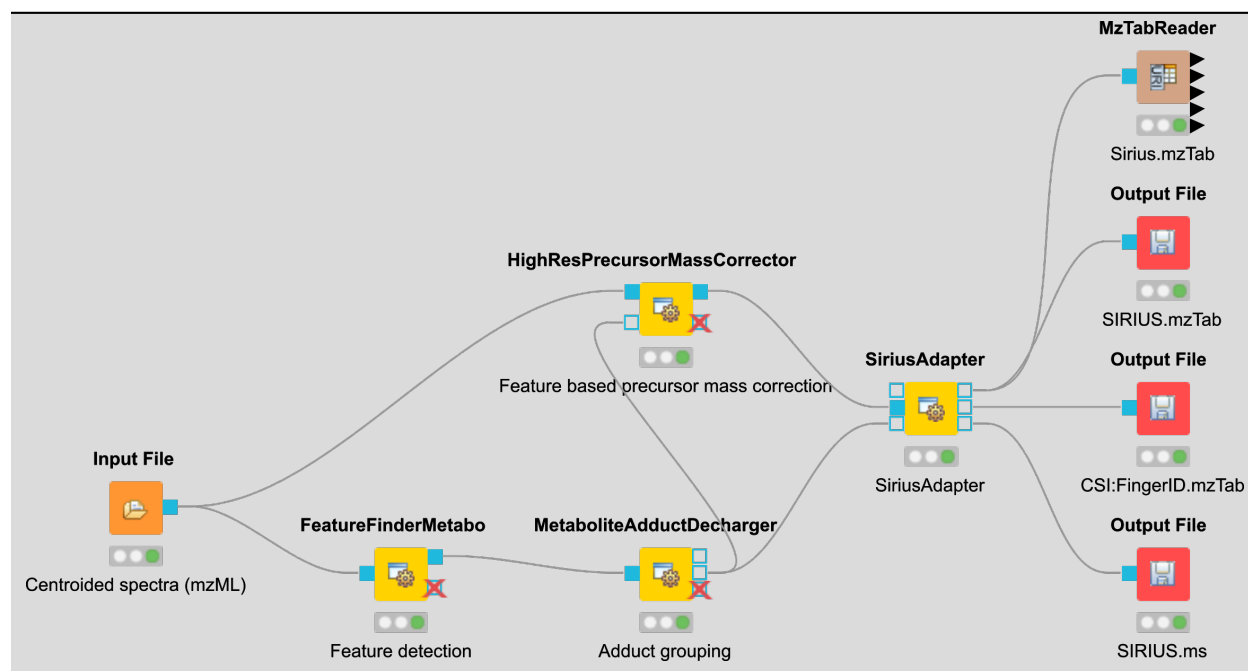


Figure 42: *De novo* identification workflow

Run the workflow and inspect the output.

The output consists of two mzTab files and an internal .ms file. One mzTab for SIRIUS and the other for the CSI:FingerID. These provide information about the chemical formula, adduct and the possible compound structure. The information is referenced to the spectrum used in the analysis. Additional information can be extracted from the **SiriusAdapter** by setting an "out_workspace_directory". Here the SIRIUS workspace will be provided after the calculation has finished. This workspace contains information about annotated fragments for each successfully explained compound.

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 `metaboliteID.knwf` 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? Note that some basic R knowledge is required to fully understand the processing in **R Snippet** nodes.

Signal processing and data preparation for identification

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

Data preparation for quantification

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(knime.in)` or `cat(knime.in)` to the Console pane.

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. 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%).

Interactive visualization

KNIME supports multiple nodes for interactive visualization with interrelated output. The nodes used in this part of the workflow exemplify this concept. They further demonstrate how figures with data dependent customization can be easily realized using basic KNIME nodes. Several simple operations are concatenated in order to enable an interactive volcano plot.

- We first log-transform fold changes and p-values in the **R Snippet** node. We then append columns noting interesting features (concerning fold change and p-value).
- With this information, we can use various Manager nodes (**Views > Property**) to emphasize interesting data points. The configuration dialogs allow us to select columns to change color, shape or size of data points dependent on the column values.
- The **Scatter Plot** node (from the **Views** repository) enables interactive visualization of the logarithmized values as a volcano plot: the log-transformed values can be chosen in the 'Column Selection' tab of the plot view. Data points can be selected in the plot and highlighted via the menu option. The highlighting transfers to all other interactive nodes connected to the same data table. In our case, selection and the highlighting will also occur in the **Interactive Table** node (from the **Views** repository).
- Output of the interactive table can then be filtered via the "HiLite" menu tab. For example, we could restrict shown rows to points highlighted in the volcano plot.

Task

Inspect the nodes of this section. Customize your visualization and possibly try to visualize other aspects of your data.

Advanced visualization

R Dependencies: This section requires that the R packages ggplot2 and ggfortify are both installed. ggplot2 is part of the KNIME R Statistics Integration (Windows Binaries) which should already be installed via the full KNIME installer, ggfortify however is not. In case that you use an R installation where one or both of them are not yet installed, add an **R Snippet** node and double-click to configure. In the R Script text editor, enter the following code:

```
#Include the next line if you also have to install ggplot2:
install.packages("ggplot2")

#Include the following lines to install ggfortify:
install.packages("ggfortify")

library(ggplot2)
library(ggfortify)
```

You can remove the:

```
install.packages
```

commands once it was successfully installed.

Even though the basic capabilities for (interactive) plots in KNIME are valuable for initial data exploration, professional looking depiction of analysis results often relies on dedicated plotting libraries. The statistics language R supports the addition of a large variety of packages, including packages providing extensive plotting capabilities. This part of the workflow shows how to use R nodes in KNIME to visualize more advanced figures. Specifically, we make use of different plotting packages to realize heatmaps.

- The used **RView (Table)** nodes combine the possibility to write R snippet code with visualization capabilities inside KNIME. Resulting images can be looked at in the output RView, or saved via the **Image Writer (Port)** node.
- The heatmap nodes make use of the **gplots** library, which is by default part of the R Windows binaries (for full KNIME version 3.1.1 or higher). We again use regular expressions to extract all measured intensity columns for plotting. For clarity, feature names are only shown in the heatmap after filtering by fold changes.

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**). For that, we use an Inner Join 2 with the **Joiner** node. In the dialog of the node, we add two entries for the Joining Columns and for the first column we pick `retention_time` from the top input (i.e. the **AccurateMassSearch** output) and `rt_cf` (the retention time of the consensus features) for the bottom input (the result from the quantification). For the second column you should choose `exp_mass_to_charge` and `mz_cf` respectively to make the joining unique. Note that the workflow needs to be executed up to the previous nodes for the possible selections of columns to appear.

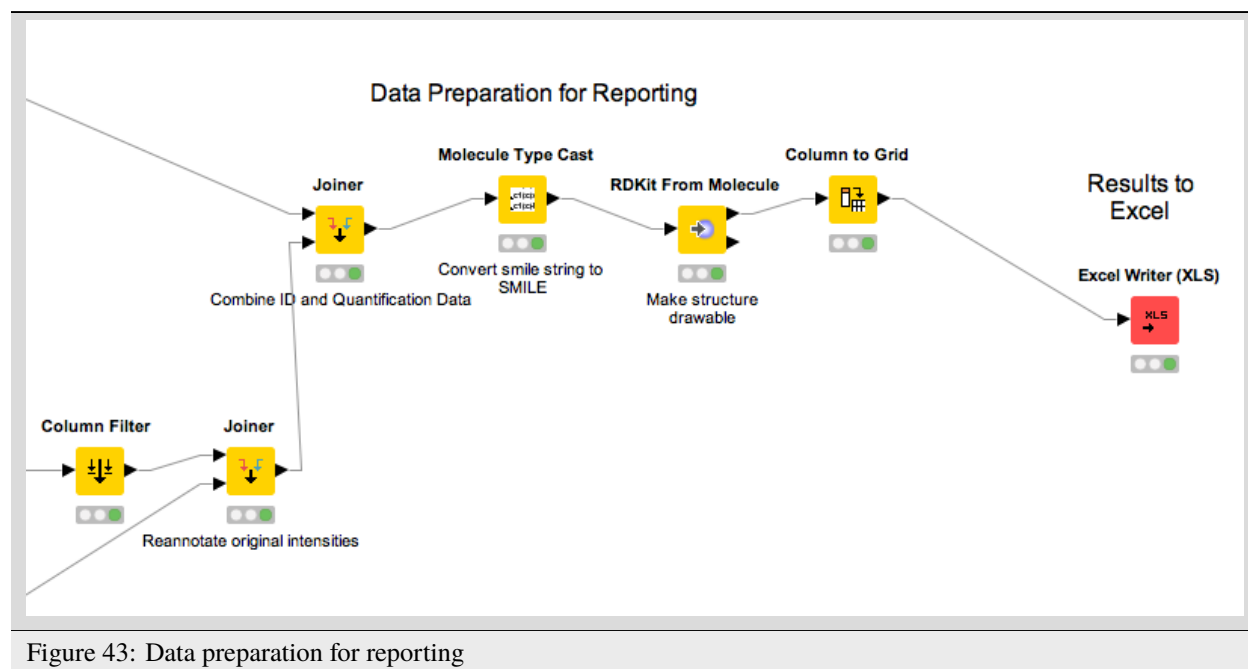


Figure 43: Data preparation for reporting

Question

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

Task

Inspect the output of the join operation after the Molecule Type Cast and RDKit molecular structure generation.

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 this end, 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 export the data to an Excel file (**XLS Writer**).

OpenSWATH

Introduction

OpenSWATH²¹ allows the analysis of LC-MS/MS DIA (data independent acquisition) data using the approach described by Gillet *et al.*²². 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.

Installation of OpenSWATH

OpenSWATH has been fully integrated since OpenMS 1.10^{Page 55, 4, Page 55, 2, 23, 24, 25}.

²¹ 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). 83, 87

²² 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. 83

²³ 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. 83

²⁴ H. L. Röst, T. Sachsenberg, S. Aiche, C. Bielow, H. Weisser, F. Aicheler, S. Andreotti, H.-c. Ehrlich, P. Gutenbrunner, E. Kenar, X. Liang, S. Nahnsen, L. Nilse, J. Pfeuffer, G. Rosenberger, M. Rurik, U. Schmitt, J. Veit, M. Walzer, D. Wojnar, W. E. Wolski, O. Schilling, J. S. Choudhary, L. Malmström, R. Aebersold, K. Reinert, and O. Kohlbacher, OpenMS: a flexible open-source software platform for mass spectrometry data analysis, *Nat. Methods* 13(9), 741–748 (sep 2016), doi:10.1038/nmeth.3959. 83

²⁵ J. Pfeuffer, T. Sachsenberg, O. Alka, M. Walzer, A. Fillbrunn, L. Nilse, O. Schilling, K. Reinert, and O. Kohlbacher, OpenMS - A platform for reproducible analysis of mass spectrometry data, *J. Biotechnol.* 261(February), 142–148 (2017), doi:10.1016/j.jbiotec.2017.05.016. 83

Installation of mProphet

mProphet²⁶ is available as standalone script in External_Tools/mProphet. R and the package MASS are further required to execute mProphet. Please obtain a version for either Windows, Mac or Linux directly from CRAN. PyProphet, a much faster reimplement of the mProphet algorithm is available from PyPI. The usage of pyprophet instead of mProphet is suggested for large-scale applications.

mProphet will be used in this tutorial.

Generating the Assay Library

Generating TraML from transition lists

OpenSWATH requires an assay library to be supplied in the TraML format²⁷. To enable manual editing of transition lists, the TOPP tool **TargetedFileConverter** is available, which uses tab separated files as input. Example datasets are provided in ExampleData/OpenSWATHAssay. Please note that the transition lists need to be named *.tsv.

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

Required Columns: PrecursorMz

The mass-to-charge (m/z) of the precursor ion. (924.539)

ProductMz

The mass-to-charge (m/z) of the product or fragment ion. (728.99)

LibraryIntensity

The relative intensity of the transition. (0.74)

NormalizedRetentionTime

The normalized retention time (or iRT)²⁸ of the peptide. (26.5)

Targeted Proteomics Columns ProteinId

A unique identifier for the protein. (AQUA4SWATH_HMLangeA)

PeptideSequence

The unmodified peptide sequence. (ADSTGTLVITDPTR)

ModifiedPeptideSequence

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

PrecursorCharge

The precursor ion charge. (2)

ProductCharge

The product ion charge. (2)

Grouping Columns: TransitionGroupId

²⁶ 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. 83

²⁷ E. W. Deutsch, M. Chambers, S. Neumann, F. Levander, P.-A. Binz, J. Shofstahl, D. S. Campbell, L. Mendoza, D. Ovelheiro, 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. 84

²⁸ 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. 84

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

TransitionId

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

Decoy

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

PeptideGroupLabel

Which label group the peptide belongs to.

DetectingTransition

Use transition for peak group detection. (1)

IdentifyingTransition

Use transition for peptidoform inference using IPF. (0)

QuantifyingTransition

Use transition to quantify peak group. (1)

For further instructions about generic transition list and assay library generation please see the following [link](#). To convert transitions lists to TraML, use the TargetedFileConverter: Please use the absolute path to your OpenMS installation.

Linux or Mac

On the Terminal:

```
TargetedFileConverter -in OpenSWATH_SGS_AssayLibrary_woDecoy.tsv -out OpenSWATH_SGS_
↳ AssayLibrary_woDecoy.TraML
```

Windows

On the TOPP command:

```
TargetedFileConverter.exe -in OpenSWATH_SGS_AssayLibrary_woDecoy.tsv -out OpenSWATH_SGS_
↳ AssayLibrary_woDecoy.TraML
```

Appending decoys to a TraML file

In addition to the target assays, OpenSWATH requires decoy assays in the library which are later used for classification and error rate estimation. For the decoy generation it is crucial that the decoys represent the targets in a realistic but unnatural manner without interfering with the targets. The methods for decoy generation implemented in OpenSWATH include 'shuffle', 'pseudo-reverse', 'reverse' and 'shift'. To append decoys to a TraML, the TOPP tool **OpenSwathDecoyGenerator** can be used: Please use the absolute path to your OpenMS installation.

Linux or Mac

On the Terminal:

```
OpenSwathDecoyGenerator -in OpenSWATH_SGS_AssayLibrary_woDecoy.TraML -out OpenSWATH_SGS_
↳ AssayLibrary.TraML -method shuffle -switchKR false
```

Windows

On the TOPP command:


```
OpenSwathDecoyGenerator.exe -in OpenSWATH_SGS_AssayLibrary_woDecoy.TraML -out OpenSWATH_
↳SGS_AssayLibrary.TraML -method shuffle -switchKR false
```

OpenSWATH KNIME

An example KNIME workflow for OpenSWATH is supplied in **Workflows** (Fig. 44). The example dataset can be used for this workflow (filenames in brackets):

1. Open **Workflows**OpenSWATH.knwf 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**. (ExampleDataOpenSWATHassayOpenSWATHiRTAssayLibrary.TraML).
3. Select the SWATH MS data in mzML format as input by double-clicking on node **Input File > SWATH-MS files**. (ExampleDataOpenSWATHdatasplitnapedroL120420x010SW-*.nf.pp.mzML).
4. Select the target peptide assay library in TraML format as input by double-clicking on node **Input Files > Assay Library**. (ExampleDataOpenSWATHassayOpenSWATHSGSAssayLibrary.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 ExampleDataOpenSWATHresult. Please use the absolute path to your R installation and the result file:

```
R --slave --args bin_dir=../../External_Tools/mProphet/ mquest=OpenSWATH_quant.tsv_
↳workflow=LABEL_FREE num_xval=5 run_log=FALSE write_classifier=1 write_all_pg=1 < ../../
↳../External_Tools/mProphet/mProphet.R
```

or for Windows:

```
"C:\Program Files\R\R-3.5.1\bin\x86\R.exe" --slave --args bin_dir=../../External_
↳Tools/mProphet/ mquest=OpenSWATH_quant.tsv 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: OpenSWATHresultmProphetxallxpeakgroups.xls with statistical information available in OpenSWATHresultmProphet.pdf.

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

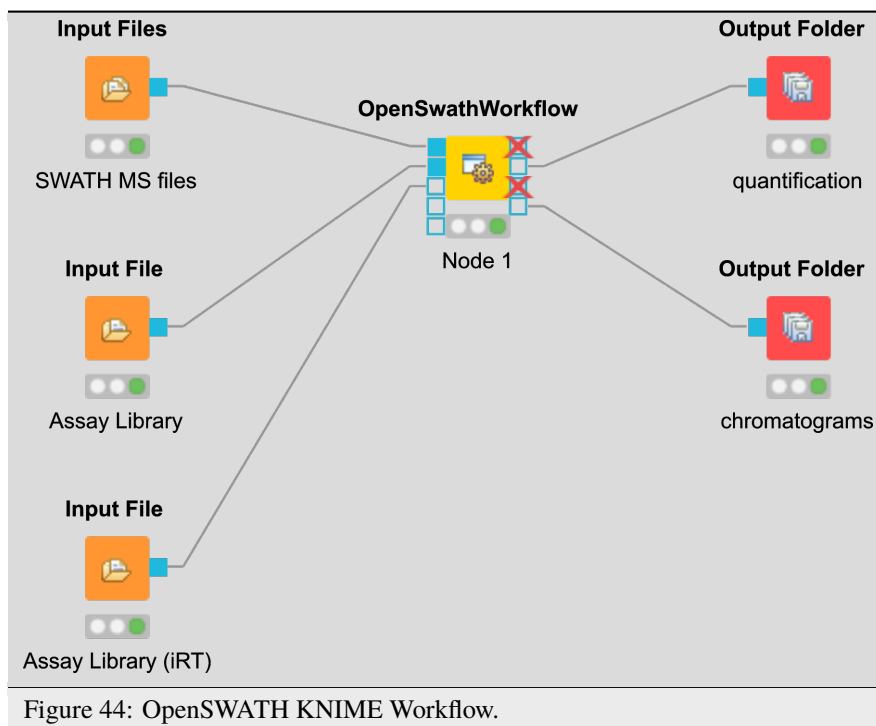


Figure 44: OpenSWATH KNIME Workflow.

Additionally, the chromatogram output (.mzML) can be visualized for inspection with TOPPView. For additional instructions on how to use pyProphet instead of mProphet please have a look at the [PyProphet Legacy Workflow](#). If you want to use the SQLite-based workflow in your lab in the future, please have a look [here](#). The SQLite-based workflow will not be part of the tutorial.

From the example dataset to real-life applications

The sample dataset used in this tutorial is part of the larger SWATH MS Gold Standard (SGS) dataset which is described in the publication of Roest *et al.*²¹. 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. Usually SWATH-MS datasets are huge with several gigabyte per run. Especially when complex samples in combination with large assay libraries are analyzed, the TOPP tool based workflow requires a lot of computational resources. Additional information and instruction can be found at the following [link](#).

OpenSWATH for Metabolomics

Introduction

We would like to present an automated DIA/SWATH analysis workflow for metabolomics, which takes advantage of experiment specific target-decoy assay library generation. This allows for targeted extraction, scoring and statistical validation of metabolomics DIA data^{29, 30}.

²⁹ H. L. Röst, G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinović, O. T. Schubert, W. Wolski, B. C. Collins, J. Malmström, L. Malmström, and R. Aebersold, OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data., Nat. Biotechnol. 32(3), 219–23 (2014), doi:10.1038/nbt.2841. 89, 90

³⁰ J. Teleman, H. L. Röst, G. Rosenberger, U. Schmitt, L. Malmström, J. Malmström, and F. Levander, DIANA-algorithmic improvements for analysis of dataindependent acquisition MS data, Bioinformatics 31(4), 555–562 (2015), arXiv: 9808008, doi:10.1093/bioinformatics/btu686. 89, 90

Workflow

The workflow follows multiple steps (see Fig. 45).

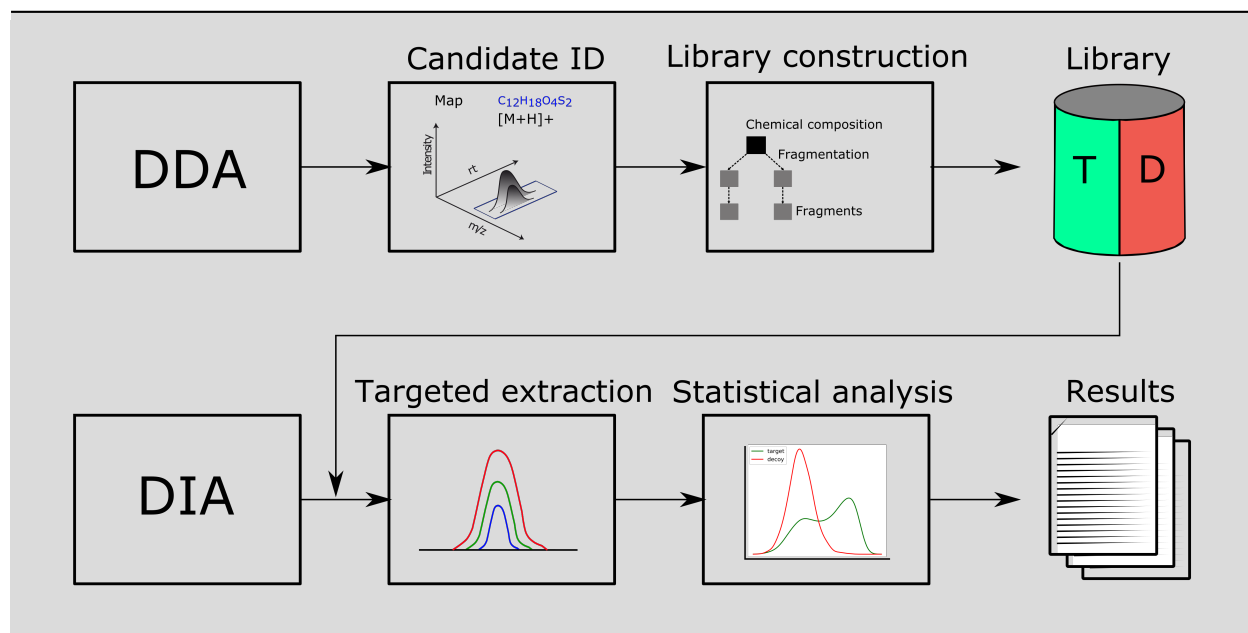


Figure 45: DIAMetAlyzer - pipeline for assay library generation and targeted analysis with statistical validation. DDA data is used for candidate identification containing feature detection, adduct grouping and accurate mass search. Library construction uses fragment annotation via compositional fragmentation trees and decoy generation using a fragmentation tree re-rooting method to create a target-decoy assay library. This library is used in a second step to analyse metabolomics DIA data performing targeted extraction, scoring and statistical validation (FDR estimation).

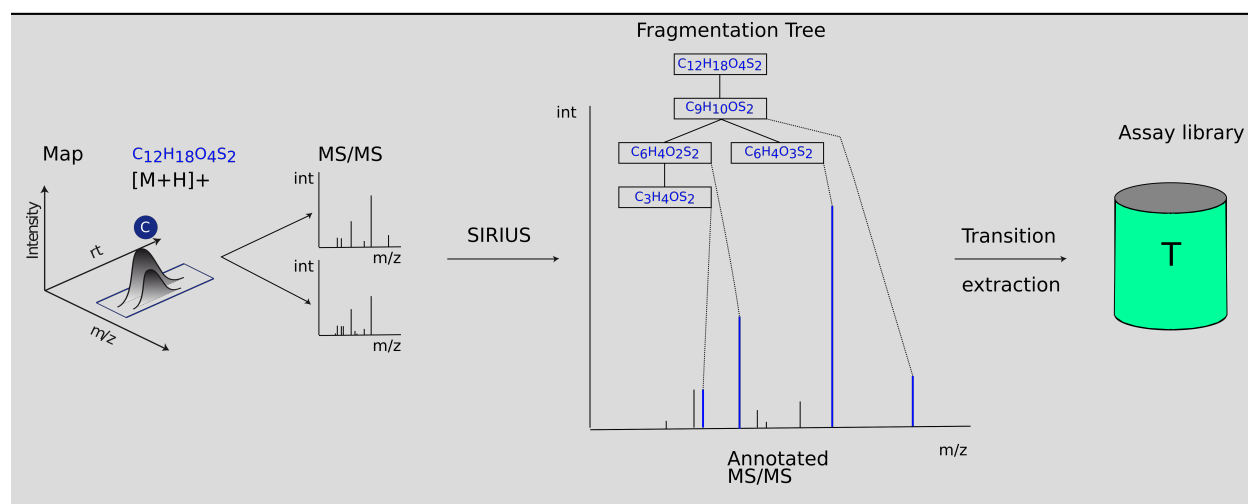


Figure 46: Assay library generation. The results of the compound identification (feature, molecular formula, adduct), with the corresponding fragment spectra for the feature, are used to perform fragment annotation via SIRIUS, using the compositional fragmentation trees. Then, the n highest intensity transitions are extracted and stored in the assay library.

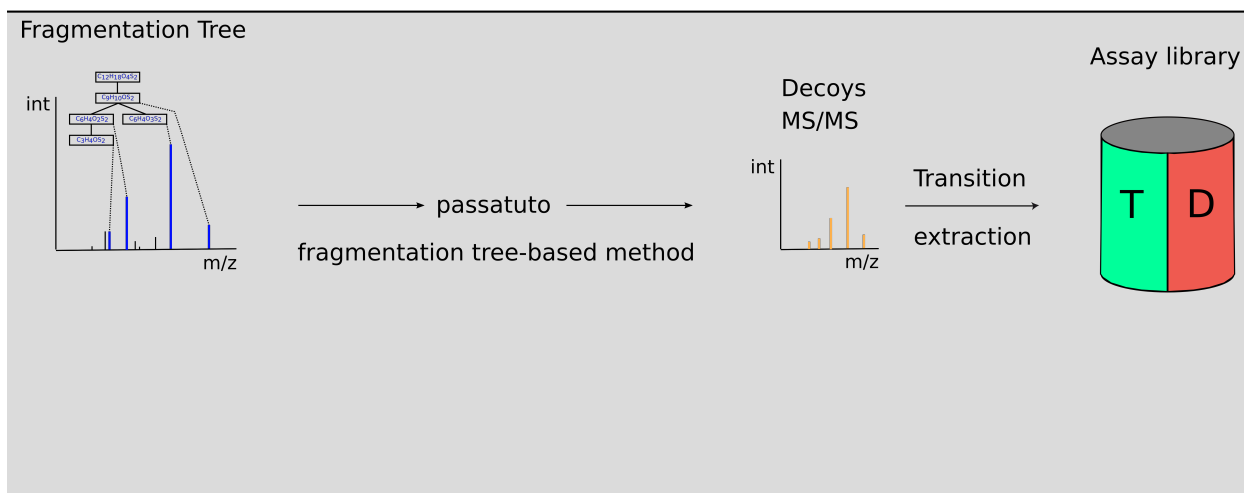


Figure 47: Decoy generation. The compositional fragmentations trees from the step above are used to run the fragmentation tree re-rooting method from Passatutto, generating a compound specific decoy MS2 spectrum. Here, the n highest intensity decoy transitions are extracted and stored in the target-decoy assay library.

- **Candidate identification** Feature detection, adduct grouping and accurate mass search are applied on DDA data.
- **Library construction** The knowledge determined from the DDA data, about compound identification, its potential adduct and the corresponding fragment spectra are used to perform fragment annotation via compositional fragmentation trees using SIRIUS 4³¹. Afterwards transitions, which are the reference of a precursor to its fragment ions are stored in a so-called assay library (Fig. 46). Assay libraries usually contain additional metadata (i.e. retention time, peak intensities). FDR estimation is based on the target-decoy approach³². For the generation of the MS2 decoys, the fragmentation tree-based rerooting method by Passatutto ensures the consistency of decoy spectra (Fig.47)³³. The target-decoy assay library is then used to analyse the SWATH data.
- **Targeted extraction** Chromatogram extraction and peak-group scoring. This step is performed using an algorithm based on OpenSWATH²⁹ for metabolomics data.
- **Statistical validation** FDR estimation uses the PyProphet algorithm^{Page 122, 30}. To prevent overfitting we chose the simpler linear model (LDA) for target-decoy discrimination in PyProphet, using MS1 and MS2 scoring with low correlated scores.

Prerequisites

Apart from the usual KNIME nodes, the workflow uses python scripting nodes. One basic requirement for the installation of python packages, in particular pyOpenMS, is a package manager for python. Using conda as an environment manager allows to specify a specific environment in the KNIME settings (**File>Preferences>KNIME>Python**).

³¹ K. Dührkop, M. Fleischauer, M. Ludwig, A. A. Aksenov, A. V. Melnik, M. Meusel, P. C. Dorrestein, J. Rousu, and S. Böcker, SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information, *Nat. Methods* 16(4), 299–302 (apr 2019), doi:10.1038/s41592-019-0344-8. 89

³² J. E. Elias and S. P. Gygi, Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry, *Nat. Methods* 4(3), 207–214 (Mar. 2007). 89

³³ K. Scheubert, F. Hufsky, D. Petras, M. Wang, L. F. Nothias, K. Dührkop, N. Bandeira, P. C. Dorrestein, and S. Böcker, Significance estimation for large scale metabolomics annotations by spectral matching, *Nat. Commun.* 8(1) (2017), doi:10.1038/s41467-017-01318-5. 89

Windows

We suggest do use a virtual environment for the Python 3 installation on windows. Here you can install miniconda and follow the further instructions.

1. Create new conda python environment.

```
conda create -n py39 python=3.9
```

2. Activate py39 environment.

```
conda activate py39
```

3. Install pip (see above).

4. On the command line:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pandas

python -m pip install -U pyprophet
python -m pip install -U pyopenms
```

macOS

We suggest do use a virtual environment for the Python 3 installation on Mac. Here you can install miniconda and follow the further instructions.

1. Create new conda python environment.

```
conda create -n py39 python=3.9
```

2. Activate py39 environment.

```
conda activate py39
```

3. On the Terminal:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pandas

python -m pip install -U pyprophet
python -m pip install -U pyopenms
```

Linux

Use your package manager apt-get or yum, where possible.

1. Install Python 3.9 (Debian: python-dev, RedHat: python-devel)
2. Install NumPy (Debian/RedHat: python-numpy).
3. Install setuptools (Debian/RedHat: python-setuptools).
4. On the Terminal:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pandas

python -m pip install -U pyprophet
python -m pip install -U pyopenms
```

Benchmark data

For the assay library construction pesticide mixes (Agilent Technologies, Waldbronn, Germany) were measured individually in solvent (DDA). Benchmark DIA samples were prepared by spiking different commercially available pesticide mixes into human plasma metabolite extracts in a 1:4 dilution series, which covers 5 orders of magnitude.

The example data can be found [here](#).

Example workflow

Example workflow for the usage of the DIAMetAlyzer Pipeline in KNIME (see Fig. 48). Inputs are the SWATH-MS data in profile mode (.mzML), a path for saving the new target-decoy assay library, the SIRIUS 4.9.0 executable, the DDA data (.mzML), custom libraries and adducts for **AccurateMassSearch**, the min/max fragment mass-to-charge to be able to restrict the mass of the transitions and the path to the PyProphet executable. The DDA is used for feature detection, adduct grouping, accurate mass search and forwarded to the **AssayGeneratorMetabo**. Here, feature mapping is performed to collect MS2 spectra that belong to a feature. All information collected before (feature, adduct, putative identification, MS2 spectra) are then internally forwarded to SIRIUS. SIRIUS is used for fragment annotation and decoy generation based on the fragmentation tree re-rooting approach. This information is then used to filter spectra/decoys based on their explained intensity (min. 85%). Afterwards internal feature linking is performed which is most important for untargeted experiments using a lot of DDA data to construct the library. The constructed target-decoy assay library is processed with the SWATH-MS data in OpenSWATH. The results are used by PyProphet for scoring and output a list of metabolites with their respective q-value and quantitative information.

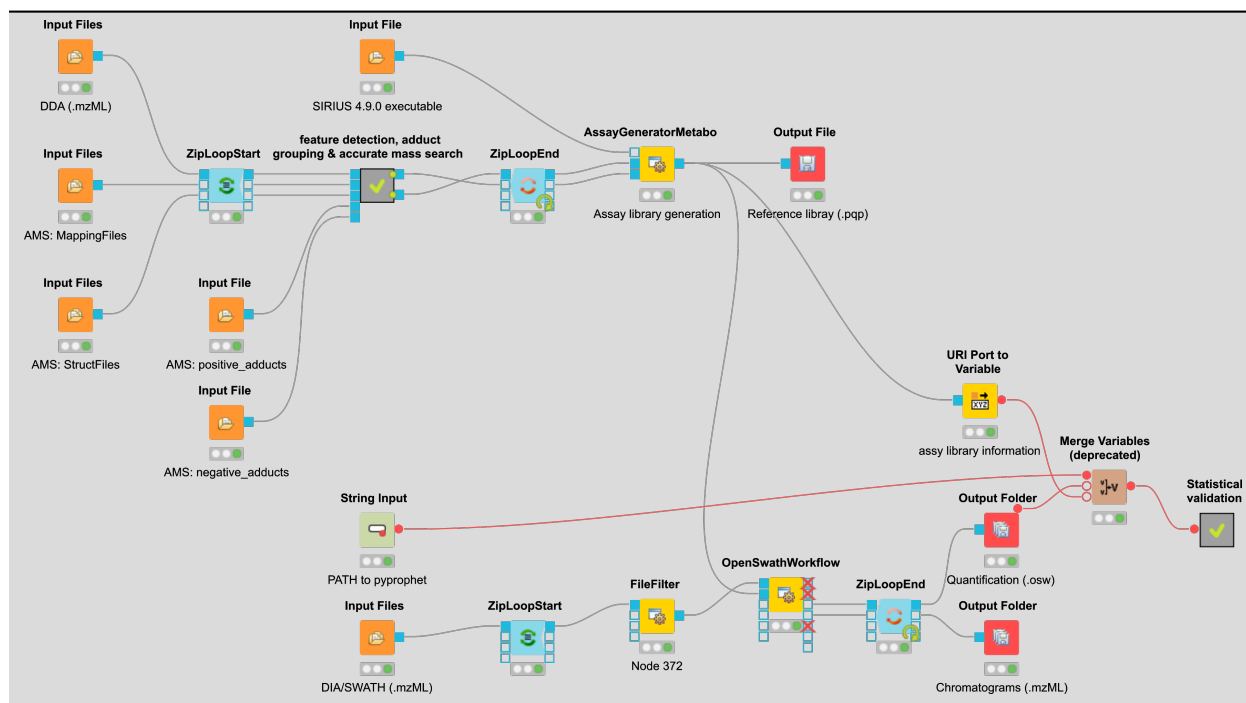


Figure 48: Example workflow for the usage of the DIAMetAlyzer Pipeline in KNIME.

Run the workflow

These steps need to be followed to run the workflow successfully:

- Add DDA Input Files (.mzML).
- Specify SIRIUS 4.9.0 executable.
- Specify library files (mapping, struct) for **AccurateMassSearch**.
- Add positive/negative adducts lists for **AccurateMassSearch**.
- Supply an output path for the SIRIUS workspace in the **AssayGeneratorMetabo**.
- Specify additional paths and variables, such as an output path for the target-decoy assay library and a path to the pyprophet installation as well as decoy fragment mz filter (min/max).
- Input DIA/SWATH files (.mzML).
- Specify output path in the output folders.

You can now run the workflow.

Important parameters

Please have a look at the most important parameters, which should be tweaked to fit your data. In general, OpenMS has a lot of room for parameter optimization to best fit your chromatography and instrumental settings.

FeatureFinderMetabo

parameter	explanation
<i>noise_threshold_int</i>	Intensity threshold below which peaks are regarded as noise.
<i>chrom_fwhm</i>	Expected chromatographic peak width (in seconds).
<i>mass_error_ppm</i>	Allowed mass deviation (in ppm).

MetaboliteAdductDecharger

parameter	explanation
<i>mass_max_diff</i>	Maximum allowed mass tolerance per feature...
<i>potential_adducts</i>	Adducts used to explain mass differences - These should fit to the adduct list specified for AccurateMassSearch.

AccurateMassSearch

parameter	explanation
<i>mass_error_value</i>	Tolerance allowed for accurate mass search.
<i>ionization_mode</i>	Positive or negative ionization mode.

AssayGeneratorMetabo

parameter	explanation
<i>min_transitions</i>	Minimal number of transitions (3).
<i>max_transitions</i>	Maximal number of transitions (3).
min_fragment_mz	Minimal m/z of a fragment ion chosen as a transition
max_fragment_mz	Maximal m/z of a fragment ion chosen as a transition
<i>transitions_threshold</i>	Further transitions need at least x% of the maximum intensity.
fragment_annotation_score	Filters annotations based on the explained intensity of the peaks in a spectrum (0.8).
SIRIUS (internal):	
<i>out_workspace_direct</i>	Output directory for SIRIUS workspace (Fragmentation Trees).
<i>filter_by_num_masstrac</i>	Features have to have at least x MassTraces. To use this parameter <i>feature_only</i> is necessary.
<i>precursor_mass_tolerance</i>	Tolerance window for precursor selection (Feature selection in regard to the precursor).
<i>precursor_rt_tolerance</i>	Tolerance allowed for matching MS2 spectra depending on the feature size (should be around the FWHM of the chromatograms).
<i>profile_elements</i>	Specify the used analysis profile (e.g. qtof).
	Allowed elements for assessing the putative sumformula (e.g. CHNOP[5]S[8]Cl[1]). Elements found in the isotopic pattern are added automatically, but can be specified nonetheless.
Feature linking (internal):	
ambiguity_resolution_mz_tolerance	M/z tolerance for the resolution of identification ambiguity over multiple files - Feature linking m/z tolerance.
ambiguity_resolution_rt_tolerance	RT tolerance in seconds for the resolution of identification ambiguity over multiple files - Feature linking m/z tolerance.
total_occurrence_filter	Filter compound based on total occurrence in analysed samples.

In case of the **total_occurrence_filter** the value to chose depends on the analysis strategy used. In the instance you are using only identified compounds (**use_known_unknowns**= false) - it will filter based on identified features. This means that even if the feature was detected in e.g. 50% of all samples it might be only identified correctly by accurate mass search in 20% of all samples. Using a **total_occurrence_filter** this specific feature would still be filtered out due to less identifications.

OpenSWATH

parameter	explanation
<i>rt_extraction_window</i>	Extract x seconds around this value.
<i>rt_normalization_factor</i>	Please use the range of your gradient e.g. 950 seconds.

If you are analysing a lot of big DIA mzML files 3-20GB per File, it makes sense to change how OpenSWATH processes the spectra.

parameter	explanation
<i>readOptions</i>	Set cacheWorkingInMemory - will cache the files to disk and read SWATH-by-SWATH into memory
<i>tempDirectory</i>	Set a directory, where cached mzMLs are stored (be aware that his directory can be quite huge depending on the data).

In the workflow pyprophet is called after OpenSWATH, it merges the result files, which allows to get enough data for the model training.

```
pyprophet merge --template path_to_target-decoy_assay_library.pqp --out merged.osw, → ./
→ *.osw
```

Afterwards, the results are scored using the MS1 and MS2 levels and filter for metabolomics scores, which have a low correlation.

```
pyprophet score --in merged.osw --out scored.osw --level ms1ms2 --ss_main_score, → "var_
→ isotope_correlation_score" --ss_score_filter metabolomics
```

Export the non filtered results:

```
pyprophet export-compound --in scored.osw --out scored + "_pyprophet_nofilter_ms1ms2.
→ tsv" --max_rs_peakgroup_qvalue 1000.0
```

Please see the workflow for actual parameter values used for the benchmarking dataset.

The workflow can be used without any identification (remove `AccurateMassSearch`). Here, all features (**known_unknowns**) are processed. The assay library is constructed based on the chemical composition elucidated via the fragment annotation (SIRIUS 4). It is also possible to use identified and in addition unknown (non-identified) features, by using **AccurateMassSearch** in combination with the `use_known_unknowns` in the **AssayGeneratorMetabo**.

Untargeted metabolomics preprocessing

The universal workflow for untargeted metabolomics always consists of feature detection in the individual MS sample files and their linkage to consensus features with common m/z and retention time values. In addition, there are optional steps such as adduct detection and annotation of features with associated MS2 spectra. This workflow prepares all the file necessary to do formula and structural annotations via **SiriusAdapter**. Furthermore it prepares all required files to run **GNPSExport**, which generates all files necessary to directly run **GNPS** Feature Based Molecular Networking (FBMN) and Ion Identity Molecular Networking (IIMN).

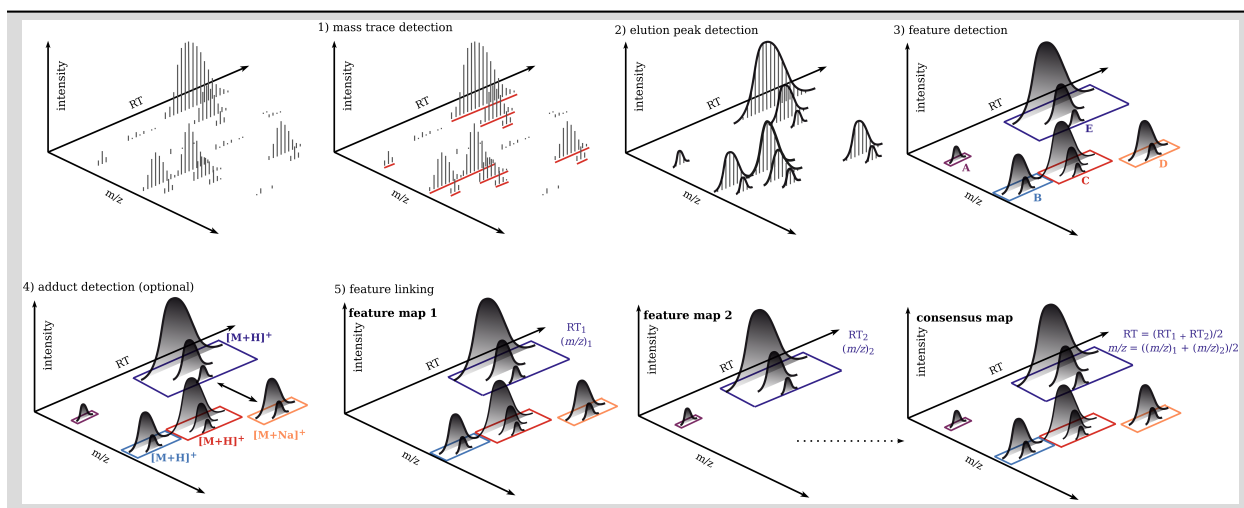


Figure 49: Metabolomics preprocessing steps

If you want to use the example data, download the files `sample1.mzML` and `sample2.mzML`.

For each `mzML` file do mass trace, elution peak and feature detection.

```
FeatureFinderMetabo -in sample1.mzML -out sample1.featureXML -algorithm:common:noise_
↳ threshold_int 10000 -algorithm:mtd:mass_error_ppm 10 -algorithm:ffm:remove_single_
↳ traces true
FeatureFinderMetabo -in sample2.mzML -out sample2.featureXML -algorithm:common:noise_
↳ threshold_int 10000 -algorithm:mtd:mass_error_ppm 10 -algorithm:ffm:remove_single_
↳ traces true
```

Align feature retention times based on the feature map with the highest number of features (reference map).

```
MapAlignerPoseClustering -in sample1.featureXML sample2.featureXML -out aligned_sample1.
↳ featureXML aligned_sample2.featureXML -trafo_out sample1.trafoXML sample2.trafoXML -
↳ algorithm:pairfinder:distance_MZ:max_difference 10.0 -algorithm:pairfinder:distance_
↳ MZ:unit ppm
```

Align `mzML` files alignment based on FeatureMap alignment (optional, only for GNPS).

```
MapRTTransformer -in sample1.mzML -out aligned_sample1.mzML -trafo_in sample1.trafoXML
MapRTTransformer -in sample2.mzML -out aligned_sample2.mzML -trafo_in sample2.trafoXML
```

Map MS2 spectra to features as `PeptideIdentification` objects (optional, only for GNPS). Requires an empty `idXML` file.

```
IDMapper -id empty.idXML -in aligned_sample1.featureXML -spectra:in aligned_sample1.mzML_
↳ -out IDmapped_sample1.featureXML
IDMapper -id empty.idXML -in aligned_sample2.featureXML -spectra:in aligned_sample2.mzML_
↳ -out IDmapped_sample2.featureXML
```

Detect adducts (optional, only for SIRIUS and GNPS Ion Identity Molecular Networking).

```
MetaboliteAdductDecharger -in IDmapped_sample1.featureXML -out_fm adducts_sample1.
↳ featureXML -algorithm:MetaboliteFeatureDeconvolution:potential_adducts "H+:0.6"
```

(continues on next page)

(continued from previous page)

```

↪ "Na:+:0.1" "NH4:+:0.1" "H-10-1:+:0.1" "H-30-2:+:0.1"
MetaboliteAdductDecharger -in IDmapped_sample2.featureXML -out_fm adducts_sample2.
↪ featureXML -algorithm:MetaboliteFeatureDeconvolution:potential_adducts "H:+:0.6"
↪ "Na:+:0.1" "NH4:+:0.1" "H-10-1:+:0.1" "H-30-2:+:0.1"

```

Link features in a ConsensusMap.

```

FeatureLinkerUnlabeledKD -in adducts_sample1.featureXML adducts_sample2.featureXML -out_
↪ Preprocessed.consensusXML -algorithm:link:rt_tol 30.0 -algorithm:link:mz_tol 10.0

```

Export table of metabolic features as tsv file including meta values (e.g. best consensus adduct ion).

```

TextExporter -in Preprocessed.consensusXML -out Features.tsv -consensus:add_metavalues

```

You can recreate this workflow in KNIME. Download the KNIME workflow here. The workflow should look like this:

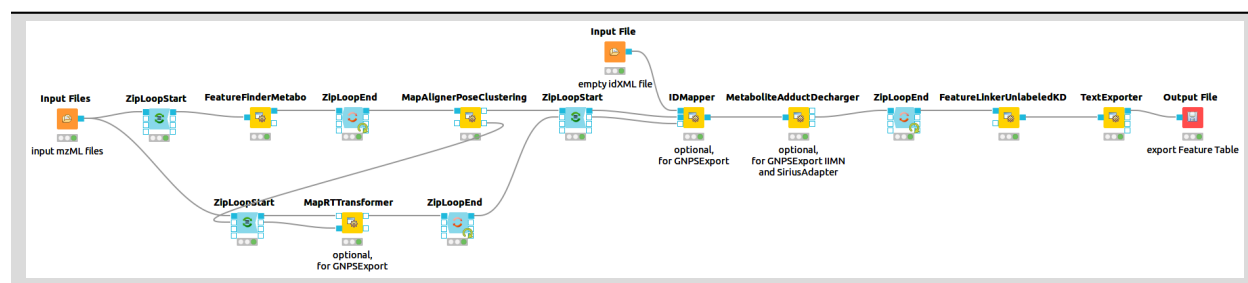


Figure 50: Metabolomics preprocessing workflow in KNIME

An introduction to pyOpenMS

Introduction

pyOpenMS provides Python bindings for a large part of the OpenMS library for mass spectrometry based proteomics and metabolomics. It thus provides access to a featurerich, open-source algorithm library for mass-spectrometry based LC-MS 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](#).

Installation

One basic requirement for the installation of python packages, in particular pyOpenMS, is a package manager for python. We provide a package for [pip](#).

Windows

1. Install Python 3.9.
2. Install NumPy.
3. Install pip (see above).
4. On the command line:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install pyopenms
```

macOS

We suggest to use a virtual environment for the Python 3 installation on Mac. Here you can install miniconda and follow the further instructions.

1. Create new conda python environment.

```
conda create -n py37 python=3.9 anaconda
```

2. Activate py37 environment.

```
source activate py37
```

3. On the Terminal:

```
pip install -U pip
pip install -U numpy
pip install pyopenms
```

Linux

Use your package manager apt-get or yum, where possible.

1. Install Python 3.9 (Debian: python-dev, RedHat: python-devel).
2. Install NumPy (Debian / RedHat: python-numpy).
3. Install setuptools (Debian / RedHat: python-setuptools).
4. On the Terminal:

```
pip install pyopenms
```

IDE with Anaconda integration

If you do not have python installed or do not want to modify your native installation, another possibility is to use an IDE (integrated development environment) with Anaconda integration. Here, we recommend [spyder](#). It comes with Anaconda, which is a package and environment manager. Thus the IDE should be able to run a specific environment independent of your systems python installation. Please execute the installer for your respective platform located in the respective directory for your platform and follow the installation instructions. After installation, the ANACONDA Navigator (Anaconda 3) should be available. Please start the application. To install pyopenms please choose the button "Environments" and click the play symbol of the base environment and "Open Terminal". Update pip and install pyopenms (MacOS, Linux):

```
pip install -U pip
pip install -U numpy
pip install -U pyopenms
```

Update pip and install pyopenms:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pyopenms
```

Install a local available package:

```
pip install numpy-1.20.0-cp37*.whl
pip install pyopenms-2.7.0-cp37*.whl
or (in case of windows)

python -m pip install -U numpy-1.20.0-cp37*.whl
python -m pip install -U pyopenms-2.7.0-cp37*.whl
```

The local available packages can be found in the directory corresponding to your operating system. Please use the absolute path to the packages for the installation.

Now launch "Spyder" (python IDE) in the home menu.

Build instructions

Instructions on how to build pyOpenMS can be found [online](#).

Scripting with pyOpenMS

A big advantage of pyOpenMS are its scripting capabilities (beyond its application in tool development). Most of the OpenMS datastructure can be accessed using [python](#). Here we would like to give some examples on how pyOpenMS can be used for simple scripting task, such as peptide mass calculation and peptide/protein digestion as well as isotope distribution calculation.

Calculation of the monoisotopic and average mass of a peptide sequence:

```
from pyopenms import *

seq = AASequence.fromString("DFPIANGER")
```

(continues on next page)

(continued from previous page)

```

mono_mass = seq.getMonoWeight(Residue.ResidueType.Full, 0)

average_mass = seq.getAverageWeight(Residue.ResidueType.Full, 0)

print("The masses of the peptide sequence " + seq.toString().decode('utf-8') + " are:")

print("mono: " + str(mono_mass))
print("average: "+ str(average_mass))

```

Enzymatic digest of a peptide/protein sequence:

```

enzyme = "Trypsin"
to_digest = AASequence.fromString("MKWVTFISLLLFSSAYSRGVFRDTHKSEIAHRFKDLGE")

after_digest = []

EnzymaticDigest = EnzymaticDigestionLogModel()

EnzymaticDigest.setEnzyme(enzyme)
EnzymaticDigest.digest(to_digest, after_digest)

print("The peptide " + to_digest.toString().decode('utf-8') + " was digested using " +
↳str(EnzymaticDigest.getEnzymeName().decode('utf-8')) + " to:")

for element in after_digest:
    print(element.toString().decode('utf-8'))

```

Use empirical formula to calculate the isotope distribution:

```

from pyopenms import *

methanol = EmpiricalFormula("CH3OH")
water = EmpiricalFormula("H2O")

wm = EmpiricalFormula(water.toString().decode('utf-8') + methanol.toString().decode('utf-
↳8'))

print(wm.toString().decode('utf-8'))
print(wm.getElementalComposition())

isotopes = wm.getIsotopeDistribution( CoarseIsotopePatternGenerator(3) )

for iso in isotopes.getContainer():
    print (iso.getMZ(), ":", iso.getIntensity())

```

For further examples and the pyOpenMS data structures please see the following [link](#).

Tool development with pyOpenMS

Scripting is one side of pyOpenMS, the other is the ability to create Tools using the C++ OpenMS library in the background. In the following section we will create a "ProteinDigester" pyOpenMS Tool. It should be able to read in a fasta file. Digest the proteins with a specific enzyme (e.g. Trypsin) and export an idXML output file. Please see ExampleDatapyopenms for code snippets.

```
usage: ProteinDigester.py [-h] [-in INFILE] [-out OUTFILE] [-enzyme ENZYME]
                        [-min_length MIN_LENGTH] [-max_length MAX_LENGTH]
                        [-missed_cleavages MISSED_CLEAVAGES]
```

ProteinDigester In silico digestion of proteins.

optional arguments:

```
-h, --help            show this help message and exit
-in INFILE            An input file containing amino acid sequences [fasta]
-out OUTFILE          Output digested sequences in idXML format [idXML]
-enzyme ENZYME        Enzyme used for digestion
-min_length MIN_LENGTH Minimum length of peptide
-max_length MAX_LENGTH Maximum length of peptide
-missed_cleavages MISSED_CLEAVAGES The number of allowed missed_
↪cleavages
```

Basics

First, your tool needs to be able to read parameters from the command line and provide a main routine. Here standard Python can be used (no pyOpenMS is required so far).

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

def main(options):
    # test parameter handling

    print(options.infile, options.outfile, options.enzyme, options.min_length, options.
↪max_length, options.missed_cleavages)

def handle_args():
    import argparse

    usage = ""

    usage += "\nProteinDigester In silico digestion of proteins."
```

(continues on next page)

(continued from previous page)

```

parser = argparse.ArgumentParser(description = usage)

parser.add_argument('-in', dest='infile', help='An input file containing amino acid_
↳sequences [fasta]')

parser.add_argument('-out', dest='outfile', help='Output digested sequences in idXML_
↳format [idXML]')

parser.add_argument('-enzyme', dest='enzyme', help='Enzyme used for digestion')

parser.add_argument('-min_length', type=int, dest='min_length', help = 'Minimum_
↳length of peptide')

parser.add_argument('-max_length', type=int, dest='max_length', help='Maximum length_
↳of peptide')

parser.add_argument('-missed_cleavages', type=int, dest='missed_cleavages', help=
↳'The number of allowed missed cleavages')

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

if __name__ == '__main__':

    options = handle_args()
    main(options)

```

Open the Anaconda Terminal and change into the ExampleDatapyopenms directory. Execute the example script.

```

python ProteinDigester_argparse.py -h
python ProteinDigester_argparse.py -in mini_example.fasta -out mini_example_out.idXML -
↳enzyme Trypsin -min_length 6 -max_length 40 -missed_cleavages 1

```

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.

OpenMS has a `ProteaseDB` class containing a list of enzymes which can be used for digestion of proteins. You can add this to the `argparse` code to be able to see the usable enzymes. From this point onward, `pyOpenMS` is required.

```

# from here pyopenms is needed
# get available enzymes from ProteaseDB

all_enzymes = []
p_db=ProteaseDB().getAllNames(all_enzymes)

# concatenate them to the enzyme argument.
parser.add_argument('-enzyme', dest='enzyme', help='Enzymes which can be used for_
↳digestion: ' + ', '.join(map(bytes.decode, all_enzymes)))

```

Loading data structures with pyOpenMS

We already scripted enzymatic digestion with the `AASequence` and `EnzymaticDigest` (see above). To make this even easier, we can use an existing class in OpenMS, called `ProteaseDigestion`.

```
# Use the ProteaseDigestion class
# set the enzyme used for digestion and the number of missed cleavages
digestor = ProteaseDigestion()
digestor.setEnzyme(options.enzyme)
digestor.setMissedCleavages(options.missed_cleavages)
# call the ProteaseDigestion::digest function
# which will return the number of discarded digestions products
# and fill the current_digest list with digested peptide sequences
digestor.digest(aaseq.fromString(fe.sequence), current_digest, options.min_length,
↳ options.max_length)
```

The next step is to use `FASTAFile` class to read the fasta input:

```
# construct a FASTAFile Object and read the input file
ff = FASTAFile()

ff.readStart(options.infile)

# construct and FASTAEntry Object
fe = FASTAEntry()

# loop over the entry in the fasta while using while
while(ff.readNext(fe)):
```

The output `idXML` needs the information about protein and peptide level, which can be saved in the `ProteinIdentification` and `PeptideIdentification` classes.

```
idxml = IdXMLFile()
idxml.store(options.outfile, protein_identifications, peptide_identifications)
```

This is the part of the program which unifies the snippets provided above. Please have a closer look how the protein and peptide datastructure is incorporated in the program.

```
def main(options):
    # read fasta file
    ff = FASTAFile()
    ff.readStart(options.infile)

    fe = FASTAEntry()

    # use ProteaseDigestion class
    digestor = ProteaseDigestion()

    digestor.setEnzyme(options.enzyme)
    digestor.setMissedCleavages(options.missed_cleavages)

    # protein and peptide datastructure
```

(continues on next page)

(continued from previous page)

```

protein_identifications = []

peptide_identifications = []
protein_identification = ProteinIdentification()

protein_identifications.append(protein_identification)
temp_pe = PeptideEvidence()

# number of dropped peptides due to length restriction
dropped_by_length = 0

while(ff.readNext(fe)):

    # construct ProteinHit and fill it with sequence information
    temp_protein_hit = ProteinHit()

    temp_protein_hit.setSequence(fe.sequence)
    temp_protein_hit.setAccession(fe.identifier)

    # save the ProteinHit in a ProteinIdentification Object

    protein_identification.insertHit(temp_protein_hit)

    # construct a PeptideHit and save the ProteinEvidence (Mapping) for the specific_
    ↪ current protein

    temp_peptide_hit = PeptideHit()
    temp_pe.setProteinAccession(fe.identifier);

    temp_peptide_hit.setPeptideEvidences([temp_pe])

    # digestion

    current_digest = []
    aaseq = AASequence()
    if (options.enzyme == "none"):

        current_digest.append(aaseq.fromString(fe.sequence))
    else:

        ↪ dropped_by_length += digester.digest(aaseq.fromString(fe.sequence), current_
        ↪ digest, options.min_length, options.max_length)

    for seq in current_digest:
        # fill the PeptideHit and PeptideIdentification datastructure

        peptide_identification = PeptideIdentification()

```

(continues on next page)

(continued from previous page)

```
temp_peptide_hit.setSequence(seq)

peptide_identification.insertHit(temp_peptide_hit)

peptide_identifications.append(peptide_identification)

print(str(dropped_by_length) + " peptides have been dropped due to the length_
↳restriction.")

idxml = IdXMLFile()
idxml.store(options.outfile, protein_identifications, peptide_identifications)
```

Putting things together

The parameter input and the functions can be used to construct the program we are looking for. If you are struggling please have a look in the example data section `ProteinDigestor.py`.

Now you can run your tool in the Anaconda Terminal `ExampleDatapyopenms`.

```
python ProteinDigestor.py -in mini_example.fasta -out mini_example_out.idXML -enzyme_
↳Trypsin -min_length 6 -max_length 40 -missed_cleavages 1
```

Bonus task

Task

Implement all other 184 TOPP tools using `pyOpenMS`.

Quality control

Introduction

In this chapter, we will build on an existing workflow with `OpenMS` / `KNIME` to add some quality control (QC). We will utilize the `qcML` tools in `OpenMS` to create a file with which we can collect different measures of quality to the mass spectrometry runs themselves and the applied analysis. The file also serves the means of visually reporting on the collected quality measures and later storage along the other analysis result files. We will, step-by-step, extend the label-free quantitation workflow from section 3 with QC functions and thereby enrich each time the report given by the `qcML` file. But first, to make sure you get the most of this tutorial section, a little primer on how we handle QC on the technical level.

QC metrics and qcML

To assert the quality of a measurement or analysis we use quality metrics. Metrics are describing a certain aspect of the measurement or analysis and can be anything from a single value, over a range of values to an image plot or other summary. Thus, qcML metric representation is divided into QC parameters (QP) and QC attachments (QA) to be able to represent all sorts of metrics on a technical level. A QP may (or may not) have a value which would equal a metric describable with a single value. If the metric is more complex and needs more than just a single value, the QP does not require the single value but rather depends on an attachment of values (QA) for full meaning. Such a QA holds the plot or the range of values in a table-like form. Like this, we can describe any metric by a QP and an optional QA. To assure a consensual meaning of the quality parameters and attachments, we created a controlled vocabulary (CV). Each entry in the CV describes a metric or part/extension thereof. We embed each parameter or attachment with one of these and by doing so, connect a meaning to the QP/QA. Like this, we later know exactly what we collected and the programs can find and connect the right dots for rendering the report or calculating new metrics automatically. You can find the constantly growing controlled vocabulary [here](#). Finally, in a qcml file, we split the metrics on a per mass-spectrometry-run base or a set of mass-spectrometry-runs respectively. Each run or set will contain its QP/QA we calculate for it, describing their quality.

Building a qcML file per run

As a start, we will build a basic qcML file for each mzML file in the label-free analysis. We are already creating the two necessary analysis files to build a basic qcML file upon each mzML file, a feature file and an identification file. We use the **QCCalculator** node from **Community > OpenMS > Utilities** where also all other QC* nodes will be found. The **QCCalculator** will create a very basic qcML file in which it will store collected and calculated quality data.

- Copy your label-free quantitation workflow into a new lfq-qc workflow and open it.
- Place the **QCCalculator** node after the **IDMapper** node. Being inside the **ZipLoop**, it will execute for each of the three mzML files the **Input** node.
- Connect the first **QCCalculator** port to the first **ZipLoopStart** outlet port, which will carry the individual mzML files.
- Connect the last's ID outlet port (**IDFilter** or the ID metanode) to the second **QCCalculator** port for the identification file.
- Finally, connect the **IDMapper** outlet to the third **QCCalculator** port for the feature file.

The created qcML files will not have much to show for, basic as they are. So we will extend them with some basic plots.

- First, we will add an 2D overview image of the given mass spectrometry run as you may know it from TOPPView. Add the **ImageCreator** node from **Community Nodes > OpenMS > Utilities**. Change the width and height parameters to 640x640 as we don't want it to be too big. Connect it to the first **ZipLoopStart** outlet port, so it will create an image file of the mzML's contained run.
- Now we have to embed this file into the qcML file, and attach it to the right **QualityParameter**. For this, place a **QCEmbedder** node behind the **ImageCreator** and connect that to its third inlet port. Connect its first inlet port to the outlet of the **QCCalculator** node to pass on the qcML file. Now change the parameter `cv_acc` to `QC:0000055` which designates the attached image to be of type `QC:0000055 - MS experiment heatmap`. Finally, change the parameter `qp_att_acc` to `QC:0000004`, to attach the image to the `QualityParameter QC:0000004 - MS acquisition result details`.
- For a reference of which CVs are already defined for qcML, have a look at the following [link](#).

There are two other basic plots which we almost always might want to look at before judging the quality of a mass spectrometry run and its identifications: the **total ion current** (TIC) and the **PSM mass error** (Mass accuracy), which we have available as pre-packaged QC metanodes.

Task

Import the workflow from WorkflowsQuality ControlQC Metanodes.zip by navigating to **File > Import KNIME Workflow...**

- Copy the **Mass accuracy** metanode into the workflow behind the **QCEmbedder** node and connect it. The qcML will be passed on and the Mass accuracy plots added. The information needed was already collected by the **QCCalculator**.
- Do the same with the **TIC** metanode so that your qcML file will get passed on and enriched on each step.

R Dependencies: This section requires that the R packages **ggplot2** and **scales** are both installed. This is the same procedure as in this section. In case that you use an R installation where one or both of them are not yet installed, open the **R Snippet** nodes inside the metanodes you just used (double-click). Edit the script in the *R Script* text editor from:

```
#install.packages("ggplot2")
#install.packages("scales")
```

to

```
install.packages("ggplot2")
install.packages("scales")
```

Press **Eval script** to execute the script.

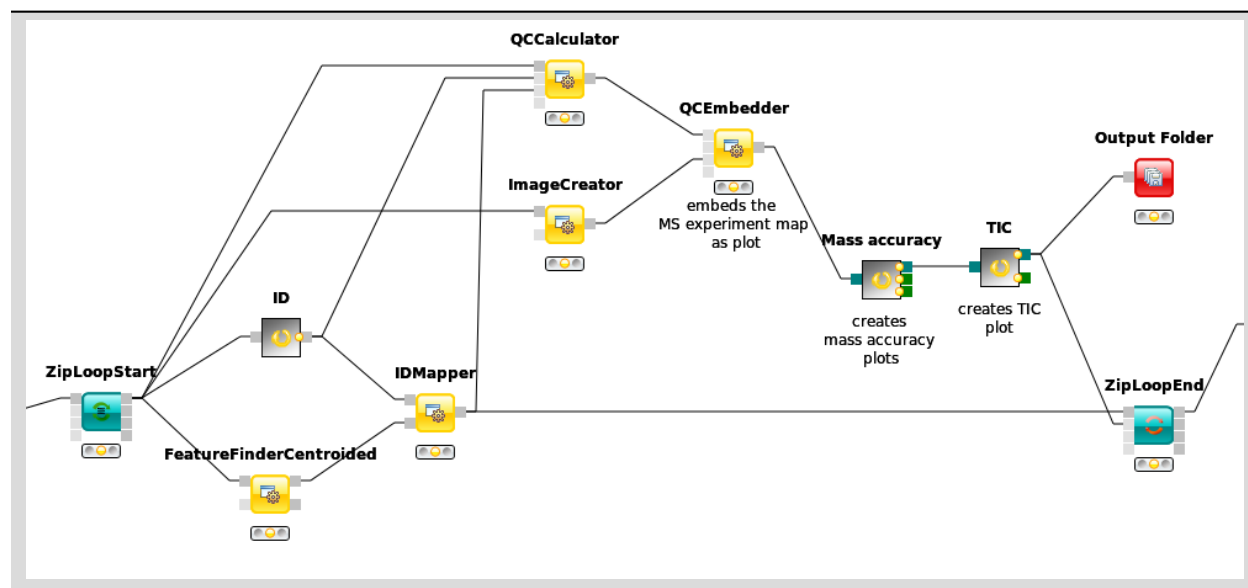


Figure 51: Basic QC setup within a LFQ workflow.

Note: To have a peek into what our qcML now looks like for one of the **ZipLoop** iterations, we can add an **Output Folder** node from **Community Nodes > GenericKnimeNodes > IO** and set its destination parameter to somewhere we want to find our intermediate qcML files in, for example **tmp > qcxfq**. If we now connect the last metanode with the Output Folder and restart the workflow, we can start inspecting the qcML files.

Task

Find your first created qcML file and open it with the browser (not IE), and the contained QC parameters will be rendered for you.

Adding brand new QC metrics

We can also add brand new QC metrics to our qcML files. Remember the **Histogram** you added inside the **ZipLoop** during the label-free quantitation section? Let's imagine for a moment this was a brand new and utterly important metric and plot for the assessment of your analyses quality. There is an easy way to integrate such new metrics into your qcMLs. Though the **Histogram** node cannot pass its plot to an image, we can do so with a **R View (table)**.

- Add an **R View (table)** next to the **IDTextReader** node and connect them.
- Edit the **R View (table)** by adding the *R Script* according to this:

```
#install.packages("ggplot2")
library("ggplot2")
ggplot(knime.in, aes(x=peptide_charge)) +

geom_histogram(binwidth=1, origin =-0.5) +
scale_x_discrete() +

ggtitle("Identified peptides charge histogram") +
ylab("Count")
```

- This will create a plot like the **Histogram** node on *peptide_charge* and pass it on as an *image*.
- Now add and connect a **Image2FilePort** node from **Community Nodes > GenericKnimeNodes > Flow** to the **R View (table)**.
- We can now use a **QCEmbedder** node like before to add our new metric plot into the qcML.
- After looking for an appropriate target from the following [link](#), we found that we can attach our plot to the *MS identification result details* by setting the parameter `qp_att_acc` to `QC:0000025`, as we are plotting the charge histogram of our identified peptides.
- To have the plot later displayed properly, we assign it the parameter `cv_acc` of `QC:0000051`, a generic plot. Also we made sure in the *R Script*, that our plot carries a caption so that we know which is which, if we had more than one new plot.
- Now we redirect the **QCEmbedders** output to the **Output Folder** from before and can have a look at how our qcML is coming along after restarting the workflow.

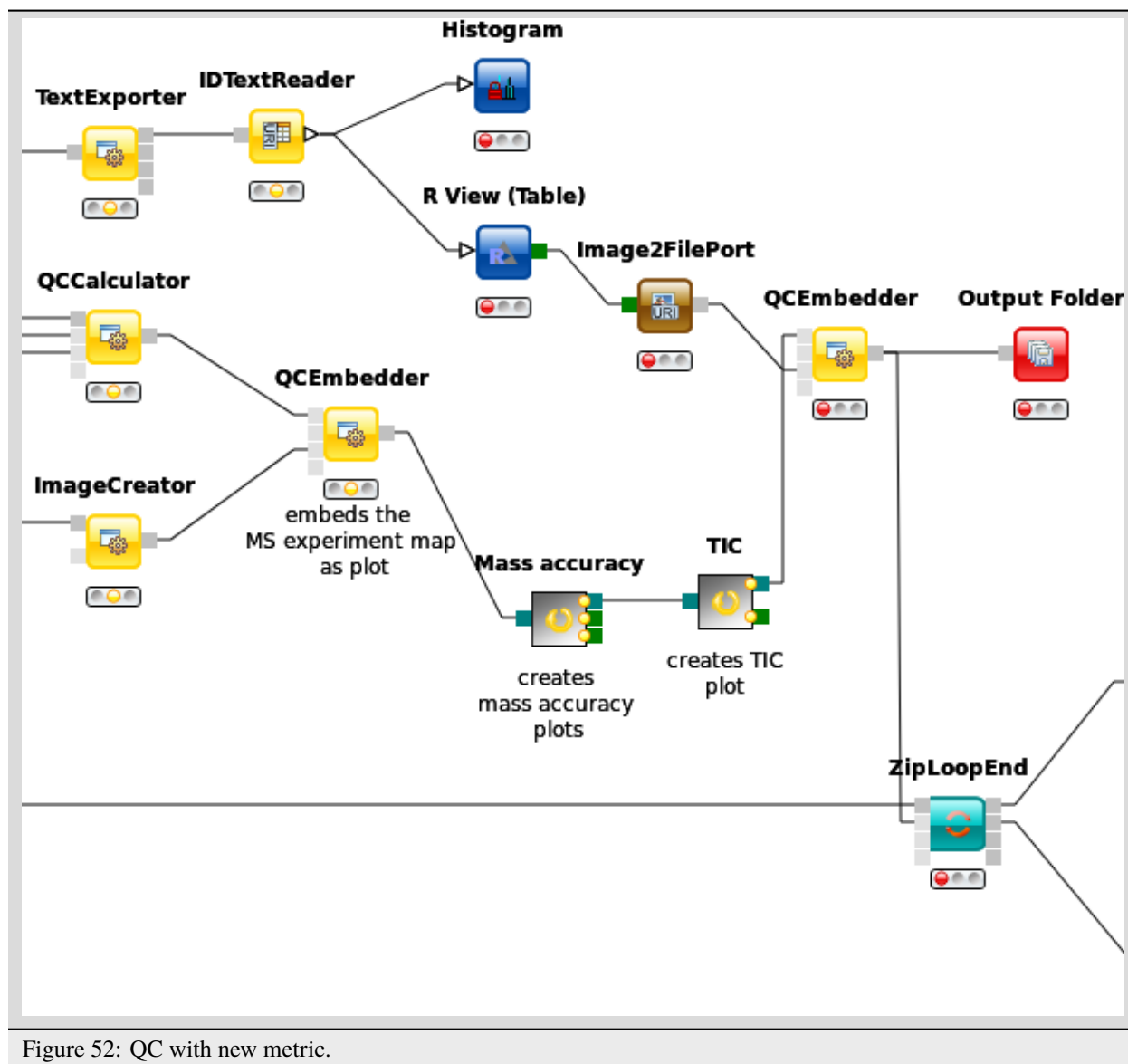


Figure 52: QC with new metric.

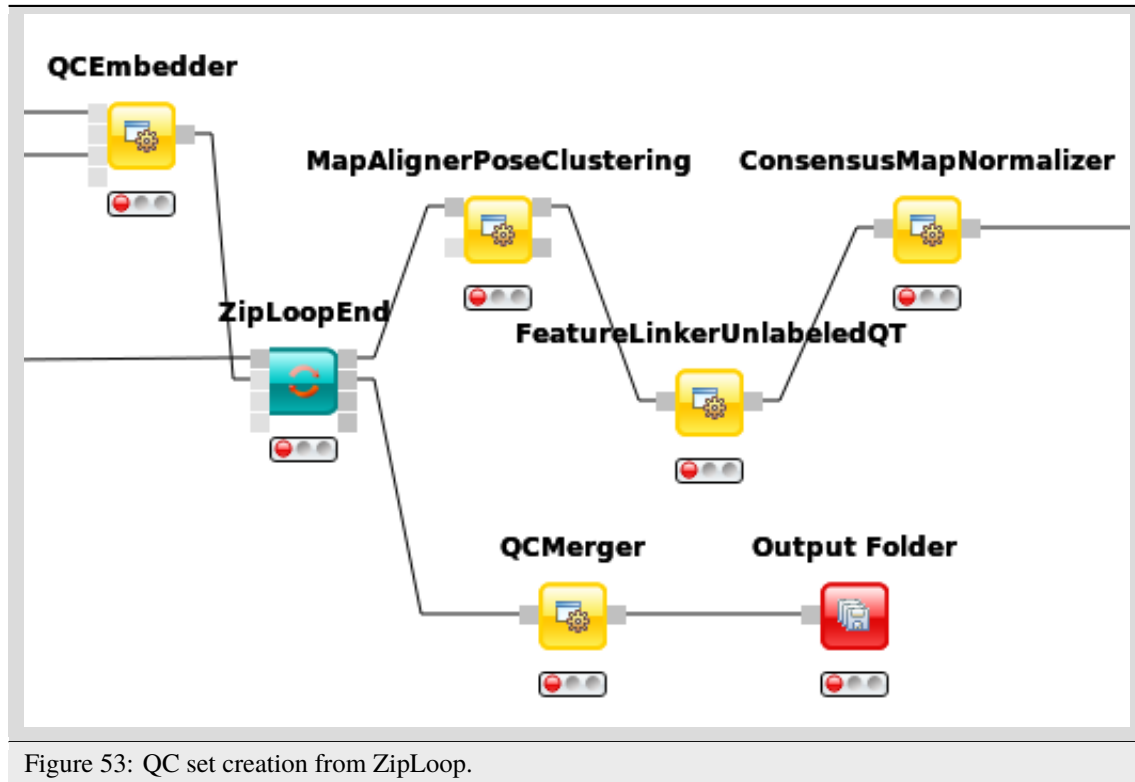
Set QC metrics

Besides monitoring the quality of each individual mass spectrometry run analysis, another capability of QC with OpenMS and qcML is to monitor the complete set. The easiest control is to compare mass spectrometry runs which should be similar, e.g. technical replicates, to spot any aberrations in the set. For this, we will first collect all created qcML files, merge them together and use the qcML onboard set QC properties to detect any outliers.

- Connect the **QCEmbedders** output from last section to the **ZipLoopEnds** second input port.
- The corresponding output port will collect all qcML files from each **ZipLoop** iteration and pass them on as a list of files.
- Now we add a **QCMerger** node after the **ZipLoopEnd** and feed it that list of qcML files. In addition, we set its parameter `setname` to give our newly created set a name - say `spikein_replicates`.
- To inspect all the QCs next to each other in that created qcML file, we have to add a new **Output Folder** to which

we can connect the **QCMerger** output.

When inspecting the set-qcML file in a browser, we will be presented another overview. After the set content listing, the basic QC parameters (like number of identifications) are each displayed in a graph. Each set member (or run) has its own section on the x-axis and each run is connected with that graph via a link in the mouseover on one of the QC parameter values.



Task

For ideas on new QC metrics and parameters, as you add them in your qcML files as generic parameters, feel free to [contact us](#), so we can include them in the CV.

Troubleshooting guide

This section will show you where you can turn to when you encounter any problems with this tutorial or with our nodes in general. Please see the [FAQ](#) first. If your problem is not listed or the proposed solution does not work, feel free to leave us a message at the means of support that you see most fit. If that is the case, please provide us with as much information as you can. In an ideal case, that would be:

- Your operating system and its version (e.g. Windows 8, Ubuntu 14.04).
- Your KNIME version (e.g. KNIME 3.1.2 full, KNIME 3.1.1 core).
- If not full: Which update site did you use for the OpenMS plugin? Trunk (nightly-builds) or Stable?
- Your OpenMS plugin version found under **Help > Install New Software > What is already installed?**
- Other installations of OpenMS on your computer (e.g. from the independent OpenMS installer, another KNIME instance etc.)

- The log of the error in KNIME and the standard output of the tool (see FAQ: How to debug).
- Your description of what you tried to do and experienced instead.

FAQ

How to debug KNIME and/or the OpenMS nodes?

- **KNIME:** Start with the normal log on the bottom right of KNIME. In general all warnings and errors will be listed there. If the output is not helpful enough, try to set the logging verbosity to the highest (DEBUG) under **Preferences > KNIME > Log file log level**.
- **OpenMS nodes:** The first step should also be the log of KNIME. Additionally, you can view the output and the errors of our tools by right-clicking on the node and selecting **View: NODENAME Std Output?error**. This shows you the output of the OpenMS executable that was called by that node. For advanced users, you can try to execute the underlying executable in your `KNIME/plugins/de.openms.platform.arch.version/payload/bin` folder, to see if the error is reproducible outside of KNIME. You can look up temporary files that are created by OpenMS nodes not connected to an Output or Viewer Node by right-clicking on a node and selecting the corresponding output view for the output you want to have a look at. The output views are located on the bottom of the menu that shows up after right-clicking. Their icon is a magnifying glass on top of a data table. The names of the output views in that menu may vary from node to node (usually a combination of "file", "out", "output" and optionally its possible extensions). For example for the Input File node you can open the information on the output files by clicking on "loaded file". In any case, a hierarchy of file descriptions will show up. If there are multiple files on that port they will be numbered (usually beginning from 0). Expand the information for the file you want to see and copy its URI (you might need to erase the "file:" prefix). Now open it with an editor of your choice. Be aware that temporary files are subject to deletion and are usually only stored as long as they are actually needed. There is also a Debug mode for the GKN nodes that keeps temporary files that can be activated under **Preferences > KNIME > Generic KNIME Nodes > Debug mode**. For the single nodes you can also increase the debug level in the configuration dialog under the advanced parameters. You can also specify a log file there, to save the log output of a specific node on your file system.

General

Q: Can I add my own modifications to the Unimod.xml?

A: Unfortunately not very easy. This is an open issue since the selections are hard-coded during creation of the tools. We included 10 places for dummy modifications that can be entered in our Unimod.xml and selected in KNIME.

Q: I have problem XYZ but it also occurs with other nodes or generally in the KNIME environment/GUI, what should I do?

A: This sounds like a general KNIME bug and we advise to search help directly at the KNIME developers. They also provide a [FAQ](#) and a [forum](#).

Q: After exporting and reading in results into a KNIME table (e.g. with a MzTabExporter and MzTabReader combination) numeric values get rounded (e.g. from scientific notation 4.5e-10 to zero) or are in a different representation than in the underlying exported file!

A: Please try a different table column renderer in KNIME. Open the table in question, right-click on the header of an affected column and select another Available Renderer by hovering and finally left-clicking.

Q: I have checked all the configurations but KNIME complains that it can not find certain output Files (FileStoreObjects).

A: Sometimes KNIME/GKN has hiccups with multiple nodes with a same name, executed at the same time in the same loop. We have seen that a simple save and restart of KNIME usually solves the problem.

Platform-specific problems

Linux

Q: Whenever I try to execute an OpenMS node I get an error similar to these:

```
/usr/lib/x86_64-linux-gnu/libgomp.so.1: version `GOMP_4.0' not found
/usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3.4.20' not found
```

A: We currently build the binaries shipped in the OpenMS KNIME plugin with gcc 4.8. We will try to extend our support for older compilers. Until then you either need to upgrade your gcc compiler or at least the library that the tool complained about or you need to build the binaries yourself (see OpenMS documentation) and replace them in your KNIME binary folder (YOURKNIMEFOLDER/plugins/de.openms.platform.architecture.version/payload/bin)

Q: Why is my configuration dialog closing right away when I double-click or try to configure it? Or why is my GUI responding so slow?

A: If you have any problems with the KNIME GUI or the opening of dialogues under Linux you might be affected by a GTK bug. See the KNIME forum (e.g. [here](#) or [here](#)) for a discussion and a possible solution. In short: set environment variable by calling `export SWT_GTK3=0` or edit `knime.ini` to make Eclipse use GTK2 by adding the following two lines:

```
-launcher.GTK_version
2
```

macOS

Q: I have problems installing RServe in my local R installation for the R KNIME Extension.

A: If you encounter linker errors while running `install.packages("Rserve")` when using an R installation from homebrew, make sure gettext is installed via homebrew and you pass flags to its lib directory. See [StackOverflow question 21370363](#).

Q: Although Ctrl + Left-click TOPPAS.app or TOPPView.app and accept the risk of a downloaded application, the icon only shortly blinks and nothing happens.

A: It seems like your OS is not able to remove the quarantine flag. If you trust us, please remove it yourself by typing the following command in your Terminal.app:

```
xattr -r -d {{ 'com.apple.quarantine /Applications/OpenMS-{{0}}'.format(version) }}
```

Windows

Q: KNIME has problems getting the requirements for some of the OpenMS nodes on Windows, what can I do?

A: Get the prerequisites installer [here](#) or install .NET3.5, .NET4 and VCRedist10.0 and 12.0 yourself.

Nodes

Q: Why is my XTandemAdapter printing empty or VERY few results, although I did not use an e-value cutoff?

A: Due to a bug in OpenMS 2.0.1, the XTandemAdapter requires a default parameter file. Give it the default configuration in YOURKNIMEFOLDER/plugins/de.openms.platform.architecture.version/payload/share/CHEMISTRY/XTandemdefaultinput.xml as a third input file. This should be resolved in newer versions though, such that it automatically uses this file if the optional inputs is empty. This should be solved in newer versions.

Q: Do MSGFPlusAdapter, LuciphorAdapter or SiriusAdapter generally behave different/unexpected?

A: These are Java processes that are started underneath. For example they can not be killed during cancellation of the node. This should not affect its performance, however. Make sure you set the Java memory parameter in these nodes to a reasonable value. Also MSGFPlus is creating several auxiliary files and accesses them during execution. Some users therefore experienced problems when executing several instances at the same time.

Sources of support

If your questions could not be answered by the FAQ, please feel free to turn to our developers via one of the following means:

- File an issue on [GitHub](#)
- Write to the [Mailing List](#)
- Open a thread on the KNIME Community Contributions [forum](#) for OpenMS

References

1.12 OpenMS in Nextflow

This page is under construction.

1.12.1 Installation of nextflow and nf-core

under construction

1.12.2 Ready-made OpenMS nextflow workflows

SCALABLE NF-CORE COMPATIBLE NEXTFLOW PIPELINES

Click on “Launch” to configure the pipeline for your data online and launch it via nextflow’s [tower app](#) (by registering a compute environment there) or by copying a configuration token for your local computer or HPC head node.

Launch “<https://nf-co.re/launch?pipeline=quantms>”

Launch “<https://nf-co.re/launch?pipeline=mhcquant>”

Launch “<https://nf-co.re/launch?pipeline=diaproteomics>”

1.12.3 Tutorial

under construction

1.13 OpenMS on Galaxy


Galaxy is an open-source web platform designed for processing and analyzing large quantities of biomedical data.


TOPP tools have been integrated into Galaxy to facilitate the creation and execution of workflows.


To use TOPP tools on Galaxy:


1. Go to the [website](#).
2. Create an account.
3. Go to **Tools** on the far left and scroll down.
4. Search for “OpenMS”.
5. You will see a list of TOPP tools.

Tools



OpenMS

 Upload Data

 Show Sections

MSFraggerAdapter Peptide Identification with MSFragger

ProteinQuantifier Compute peptide and protein abundances

OpenSwathWorkflow Complete workflow to run OpenSWATH

FileConverter Converts between different MS file formats.

FeatureFinderMultiplex Determination of peak ratios in LC-MS data

Choose one of the TOPP tools from the list. You will be able to run it in isolation or use it to create a workflow.

1.14 User Quickstart Guide

Read the User Quickstart guide to gain a brief understanding of key concepts and how to use the tools. For more in-depth information, consult [OpenMS API Reference](#).

1.14.1 What is OpenMS

OpenMS is a free, open-source C++ library with Python bindings. It is commonly used for liquid chromatography-mass spectrometry (*LC-MS*) data management and analyses. OpenMS provides an infrastructure for the rapid development of mass spectrometry related software as well as a rich toolset built on top of it. OpenMS is available under the [three clause BSD licence](#) and runs under Windows, macOS, and Linux operating systems.

1.14.2 Background

Before using OpenMS, become familiar with the following terms:

Tool and Utilities	Description
TOP-PView	A tool that is used to view and explore <i>LC-MS</i> data, alignments, groups, peptide identifications, and more.
TOP-PAS	A graphical workflow design tool that is used to create pipelines from all <i>TOPP tools</i> (and <i>UTILS</i>).
TOPP tools	A set of command line tools. Each of these command line tools is a building block of an analysis pipeline and are chained together in a way that fits the requirements of the user. The <i>TOPP tools</i> are accessible from a command prompt/shell or via <i>TOPPAS</i> .
UTILS	Besides <i>TOPP</i> , OpenMS offers range of other tools. They are not included in <i>TOPP</i> as they are not part of typical analysis pipelines. A set of command line utilities, similar to <i>TOPP tools</i> , mostly used during pipeline construction or parameter optimization.

See also:

[UTILS documentation](#)

Important: Users can now use *KNIME* in place of *TOPPAS*; the later will deprecated with no support in near future. Please find more information about using *KNIME* in KNIME tutorial.

1.14.3 How to run a Tool

A good start are the example pipelines (select **File > Open example file** within *TOPPAS*).

Alternatively, use the command line and call tools directly. In this case, you'll probably want to use some type of shell script for automation.

1.14.4 Adapt pipeline parameters

The default parameters of each tool can usually be tweaked to fit the data and improve results.

Where do you change pipeline parameters?

1. **TOPPAS:** Double-click the node of which you want to change the parameters of. A short docu for each parameter will show up once it is selected. All parameters which would be available on the command line and in the INI file are shown here as well.
2. **Command line:** Very basic parameters can be set on the command line, e.g. `FileFilter -rt 1000:2000`
.....
3. Doing 2 for all parameters would create a very long list, thus, use so-called `.ini` files to provide full parameter sets to *TOPP tools*. If no INI file is given, default parameters are used. To get a default `.ini` use

```
<tool> -write_ini <file>
```

e.g. `FileFilter -write_ini filefilter.ini`

Now, edit the INI file (which is a XML file) using the `INIFileEditor`, which is another GUI tool shipped with OpenMS and similar to the one build into *TOPPAS*.

How do I feed the INI file to a Tool?

1. **TOPPAS:** Once you changed the parameters of a node and clicked **Ok**, the parameters are in effect. Because they are part of the *TOPPAS* workflow, they are saved together with the workflow.
2. **Command line :** Supply the INI file via the `-ini` flag, `<tool> -ini <file>`
e.g. `FileFilter -ini filefilter.ini`

What parameters to set and to what value?

The answer is complex, in general, read the tool description, change the parameters and compare the results using *TOPPView* if possible. If that does not help, [contact us](#). Please include all the necessary details we need in order to help you.

1.15 OpenMS User Tutorial

1.15.1 General Remarks

- This handout will guide you through an introductory tutorial for the OpenMS/TOPP software package¹.
- OpenMS^{2,3} 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 Pipeline")⁴ can be understood as small building blocks of arbitrarily complex data analysis workflows.

¹ OpenMS, OpenMS home page [online].

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 opensource software framework for mass spectrometry., BMC bioinformatics 9(1) (2008), doi:10.1186/1471-2105-9-163. 7, 83

³ H. L. Röst, T. Sachsenberg, S. Aiche, C. Bielow, H. Weisser, F. Aicheler, S. Andreotti, H.-C. Ehrlich, P. Gutenbrunner, E. Kenar, et al., OpenMS: a flexible open-source software platform for mass spectrometry data analysis, Nature Methods 13(9), 741–748 (2016). 7

⁴ 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). 7, 83

- In order to facilitate workflow construction, OpenMS was integrated into KNIME⁵, the Konstanz Information Miner, an open-source integration platform providing a powerful and flexible workflow system combined with advanced data analytics, visualization, and report capabilities. Raw MS data as well as the results of data processing using TOPP can be visualized using TOPPView⁶.
- This tutorial was designed for use in a hands-on tutorial session but can also be worked through at home using the online resources. You will become familiar with some of the basic functionalities of OpenMS/TOPP, TOPPView, as well as KNIME and learn how to use a selection of TOPP tools used in the tutorial workflows.
- If you are attending the tutorial and received a USB stick, all sample data referenced in this tutorial can be found in the C:Example_Data folder, on the USB stick, or released online on our [Archive](#).

1.15.2 Getting Started

Installation

Before we get started, we will install OpenMS and KNIME. If you take part in a training session you will have likely received an USB stick from us that contains the required data and software. If we provide laptops with the software you may of course skip the installation process and continue reading the next section. If you are doing this tutorial online, choose online in the following tab(s).

Online

If you are working through this tutorial at home/online, proceed with the following steps:

- Download and install OpenMS using the installation instructions for your operating system:
 - [GNU/Linux](#)
 - [macOS](#)
 - [Windows](#)
- Download and install [KNIME](#)

USB Stick

Please choose the directory that matches your operating system and execute the installer.

For Windows, you run:

- The OpenMS installer: WindowsOpenMS-3.0.0-Win64.exe
- The KNIME installer: WindowsKNIME-4.7.2-Installer-64bit.exe

On macOS, you run:

- The OpenMS installer: MacOpenMS-3.0.0-macOS.dmg
- The KNIME installer: Macknime_3.0.0.app.macosx.cocoa.x86_64.dmg

On Linux, you can extract KNIME to a folder of your choice and for TOPPView you need to install OpenMS via your package manager or build it on your own with our [build instructions](#).

⁵ M. R. Berthold, N. Cebon, 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 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. 7

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 MSConvert, which is part of a ProteoWizard installation. All files used in this tutorial have already been converted to mzML by us, so you do not need to perform the data conversion yourself. However, we provide a small raw file so you can try the important step of raw data conversion for yourself.

Note: The OpenMS installation package for Windows automatically installs ProteoWizard, so you do not need to download and install it separately. Due to restrictions from the instrument vendors, file format conversion for most formats is only possible on Windows systems. In practice, performing the conversion to mzML on the acquisition PC connected to the instrument is usually the most convenient option.

To convert raw data to mzML using ProteoWizard you can either use MSConvertGUI (a graphical user interface) or `msconvert` (a simple command line tool).

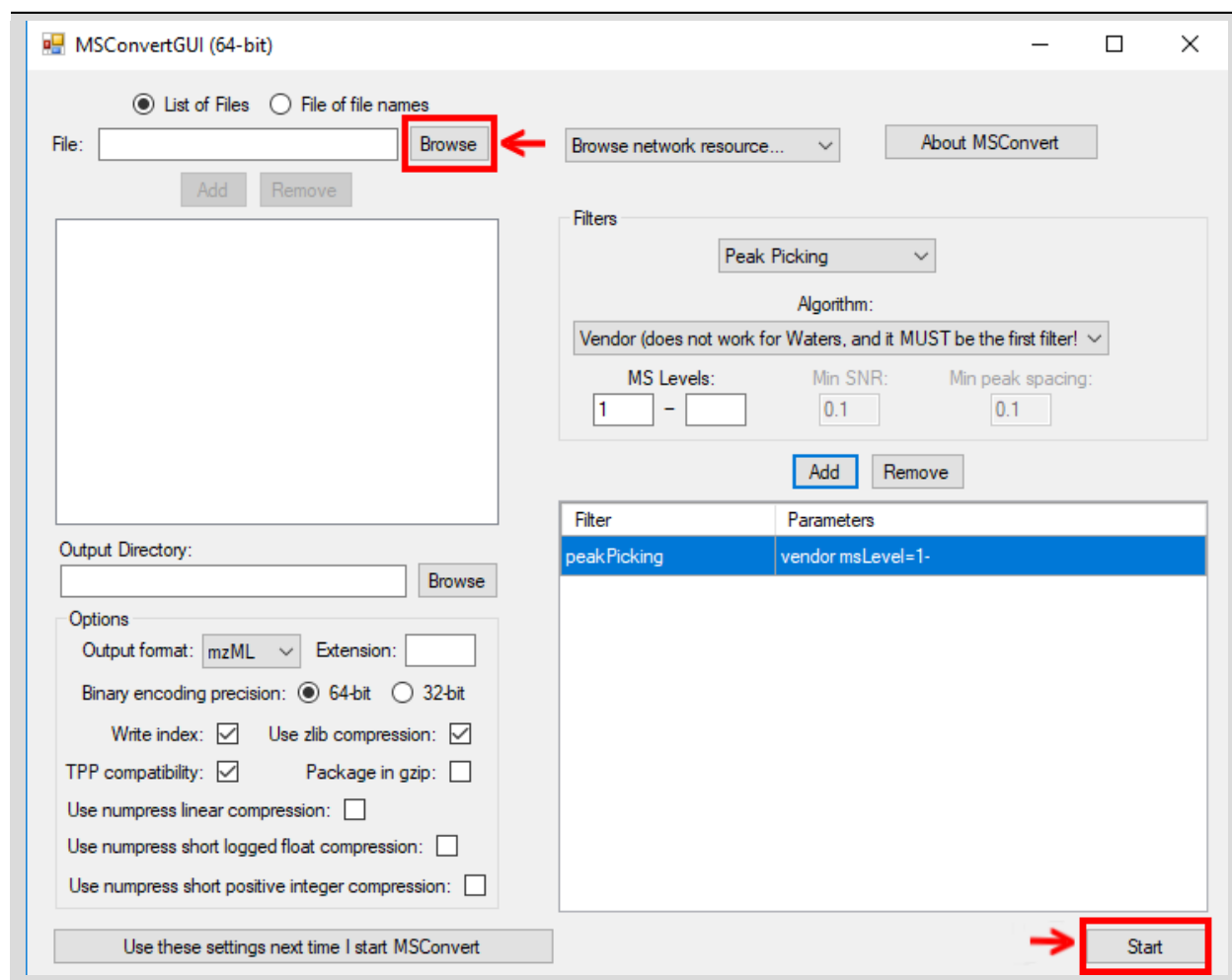


Figure 1: MSConvertGUI (part of ProteoWizard), allows converting raw files to mzML. Select the raw files you want to convert by clicking on the browse button and then on Add. Default parameters can usually be kept as-is. To reduce the initial data size, make sure that the `peakPicking` filter (converts profile data to centroided data (see Fig. 2)) is listed, enabled (true) and applied to all MS levels (parameter "1-"). Start the conversion process by clicking on the Start button.

Both tools are available in: C:\Program Files\OpenMS-3.0.0\share\OpenMS\THIRDPARTY\pwiz-bin.

You can find a small RAW file on the USB stick C:\Example_Data\Introduction\datasets\raw.

MSConvertGUI

MSConvertGUI (see Fig. 1) exposes the main parameters for data conversion in a convenient graphical user interface.

msconvert

The `msconvert` command line tool has no graphical user interface but offers more options than the application MSConvertGUI. Additionally, since it can be used within a batch script, it allows converting large numbers of files and can be much more easily automatized. To convert and pick the file `raw_data_file.RAW` you may write:

```
msconvert raw_data_file.RAW --filter "peakPicking true 1-"
```

in your command line.

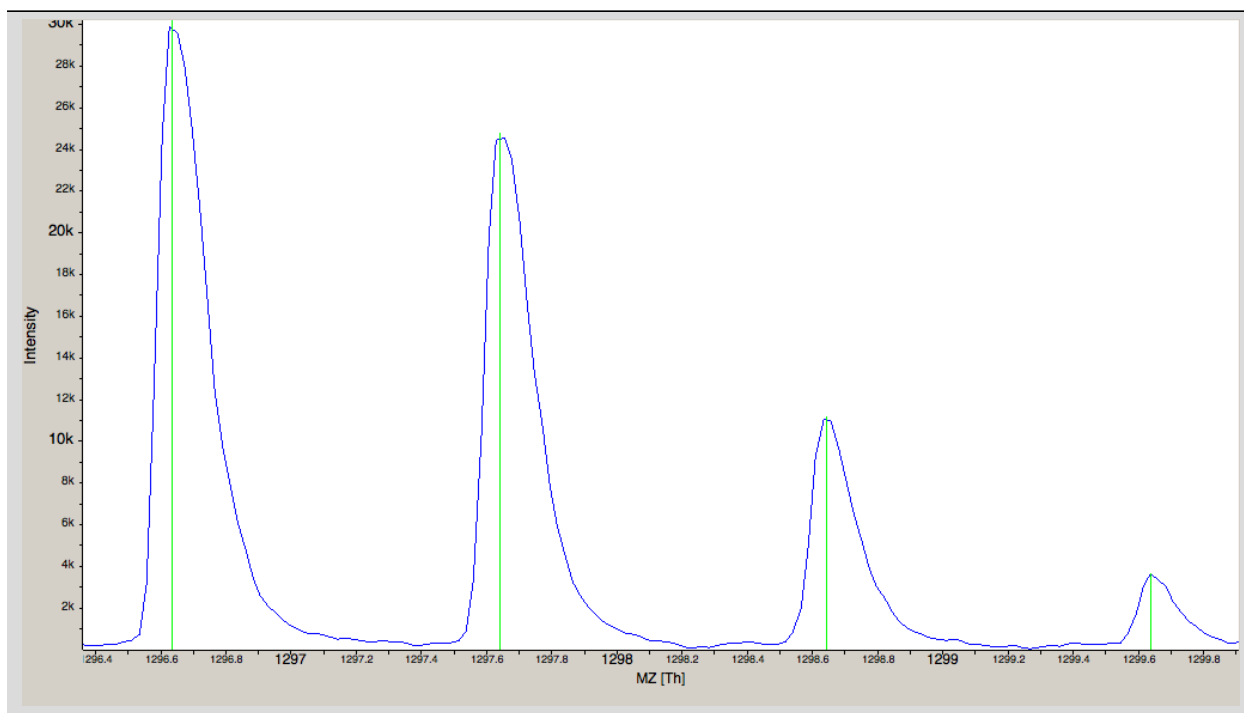


Figure 2: The amount of data in a spectra is reduced by peak picking. Here a profile spectrum (blue) is converted to centroided data (green). Most algorithms from this point on will work with centroided data.

To convert all RAW files in a folder may write:

```
msconvert *.RAW -o my_output_dir
```

Note: To display all options you may type `msconvert --help`. Additional information is available on the ProteoWizard web page.

ThermoRawFileParser

Recently the open-source platform independent ThermoRawFileParser tool has been developed. While Proteowizard and MSConvert are only available for Windows systems this new tool allows to also convert raw data on Mac or Linux.

Note: To learn more about the ThermoRawFileParser and how to use it in KNIME see A minimal workflow.

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. Note that conceptually, there are no differences in visualizing metabolomic or proteomic data. Here, we inspect a simple proteomic measurement:

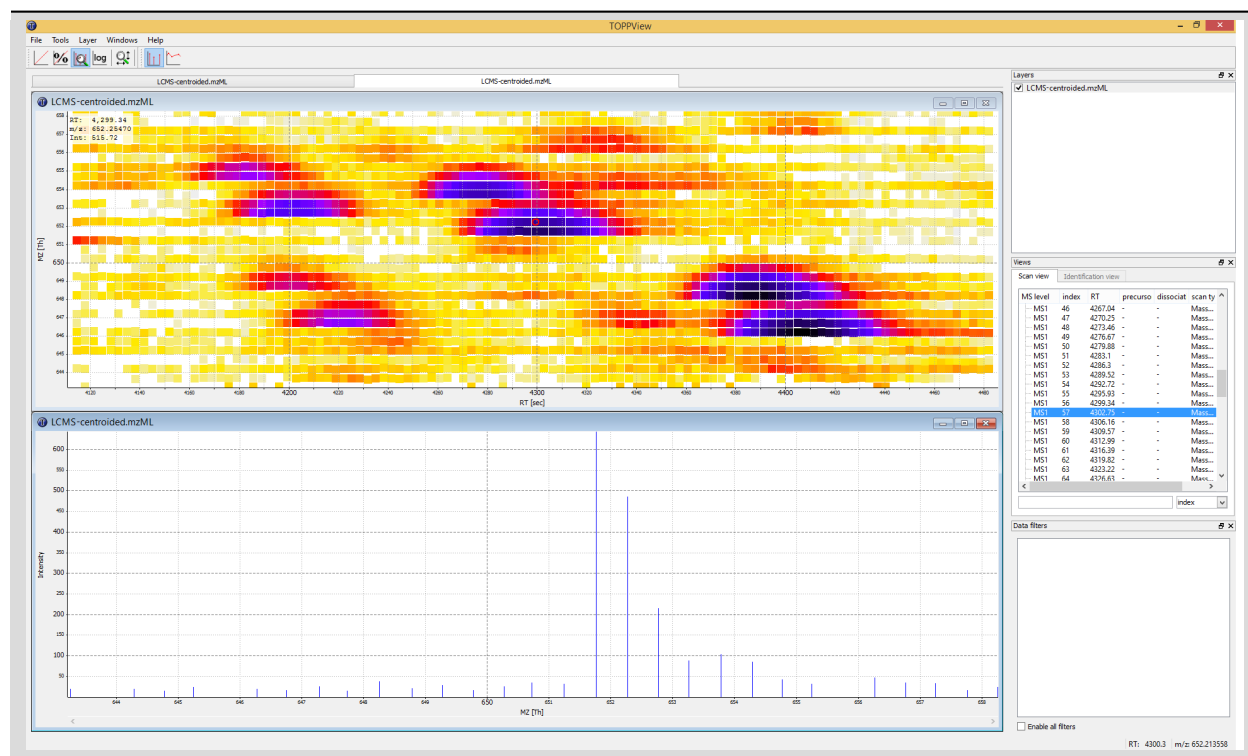


Figure 3: TOPPView, the graphical application for viewing mass spectra and analysis results. Top window shows a small region of a peak map. In this 2D representation of the measured spectra, signals of eluting peptides are colored according to the raw peak intensities. The lower window displays an extracted spectrum (=scan) from the peak map. On the right side, the list of spectra can be browsed.

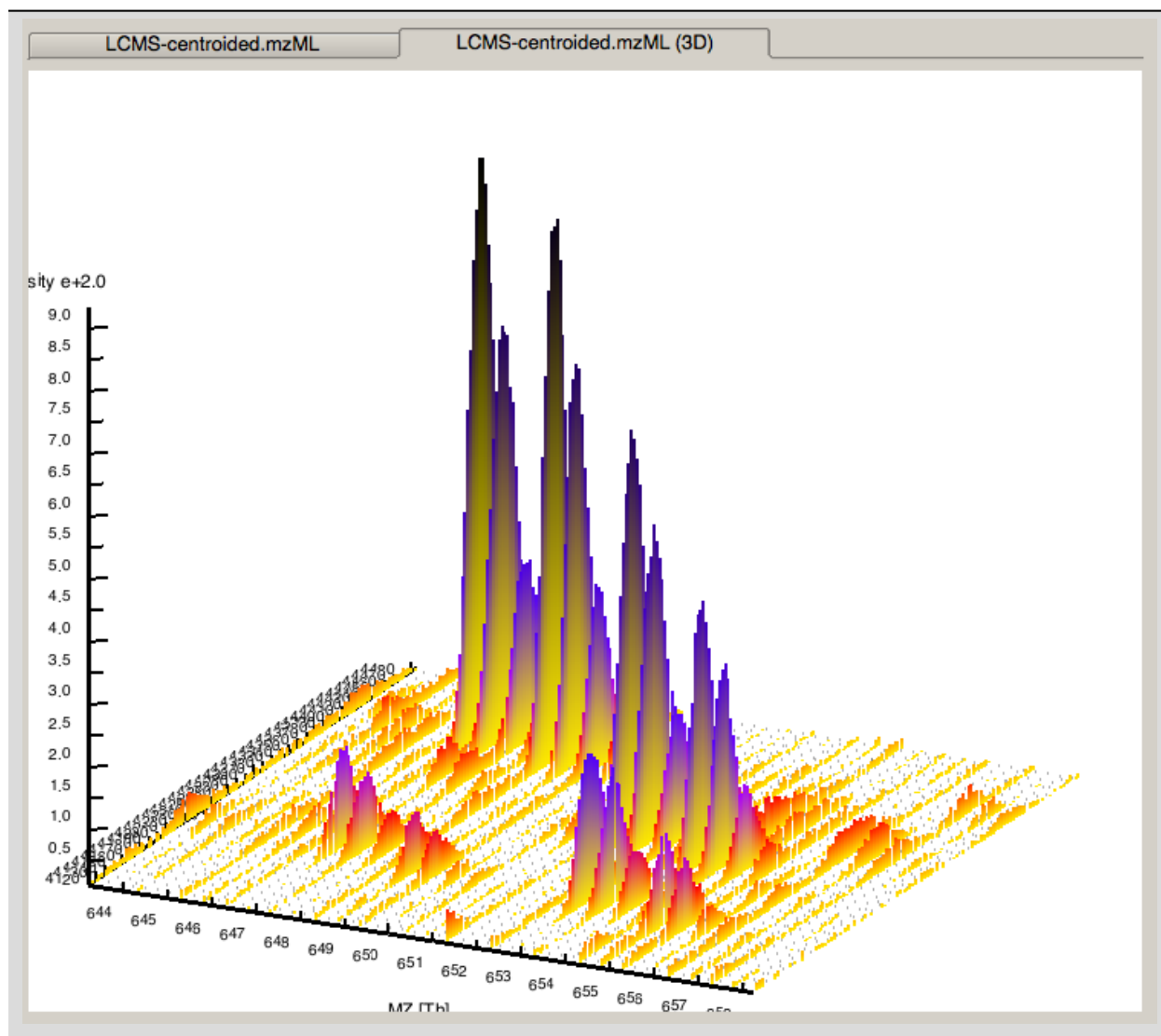


Figure 4: 3D representation of the measured spectra, signals of eluting peptides are colored according to the raw peak intensities.

- Start TOPPView (see Windows' Start-Menu or ApplicationsOpenMS-3.0.0 on macOS)
- Go to **File > Open File**, navigate to the directory where you copied the contents of the USB stick to, and select `Example_DataIntroductiondatasetssmallvelos005614.mzML`. This file contains only a reduced LC-MS map of a label-free proteomic platelet measurement recorded on an Orbitrap velos. The other two mzML files contain technical replicates of this experiment. First, we want to obtain a global view on the whole LC-MS map - the default option Map view 2D is the correct one and we can click the Ok button.
- Play around.
- Three basic modes allow you to interact with the displayed data: scrolling, zooming and measuring:
 - **Scroll mode**
 - * Is activated by default (though each loaded spectra file is displayed zoomed out first, so you do not need to scroll).
 - * Allows you to browse your data by moving around in RT and m/z range.

- * When zoomed in, you can scroll through the spectra. Click-drag on the current view.
- * Arrow keys can be used to scroll the view as well.

– Zoom mode

- * 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 zoom in and out.
- * 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).
- * Pressing backspace ← zooms out to show the full LC-MS map (and also resets the zoom history).

– Measure mode

- * It is activated using the Shift 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 mode** and examine your data in 3D mode (see Fig. 4).
 - Go back to the 2D view. In 2D mode, visualize your data in different intensity normalization modes, use linear, percentage, snap and log-view (icons on the upper left tool bar). You can hover over the icons for additional information.

Note: On macOS, 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.

Dependent on your data MS/MS spectra can be visualized as well (see Fig.5). You can do so, by double-click on the MS/MS spectrum shown in scan view

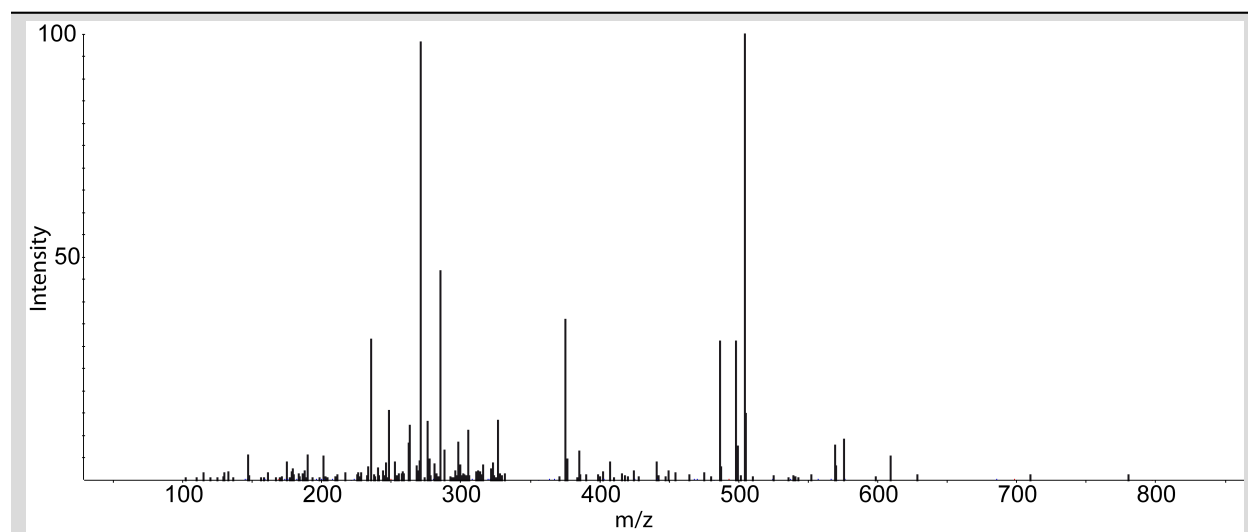


Figure 5: MS/MS spectrum

Introduction to KNIME/OpenMS

Using OpenMS in combination with KNIME, you can create, edit, open, save, and run workflows that combine 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, to make it more difficult to create an invalid workflow. Throughout most parts of this tutorial, you will use KNIME to create and execute workflows. The first step is to become familiar with KNIME. Additional information on the basic usage of KNIME can be found on the [KNIME Getting Started page](#). However, the most important concepts will also be reviewed in this tutorial.

Plugin and dependency

Before we can start with the tutorial, we need to install all the required extensions for KNIME. Since KNIME 3.2.1, the program automatically detects missing plugins when you open a workflow but to make sure that the right source for the OpenMS plugin is chosen, please follow the instructions here.

Required KNIME plugins

First, we install some additional extensions that are required by our OpenMS nodes or used in the Tutorials for downstream processing, visualization or reporting.

1. In KNIME, click on **Help > Install New Software**.
2. From the ‘**Work with:**’ drop-down list, select the *update site* ‘KNIME 4.6 - <https://update.knime.com/community-contributions/trusted/4.6>’
3. Now select the following KNIME core plugins from the KNIME & Extensions category
 - KNIME Base Chemistry Types & Nodes
 - KNIME Chemistry Add-Ons
 - KNIME Interactive R Statistics Integration
 - KNIME Report Designer
 - KNIME SVG Support
4. Click on **Next** and follow the instructions (you may but don’t need to restart KNIME now).
5. Click again on **Help > Install New Software**
6. From the ‘**Work with:**’ drop-down list, select the *update site* ‘KNIME Community Extensions (Trusted) - <https://update.knime.com/community-contributions/trusted/4.6>’
7. Now select the following plugin from the “KNIME Community Contributions - Cheminformatics” category
 - RDKit KNIME integration
8. Click on **Next** and follow the instructions and after a restart of KNIME the dependencies will be installed.

R programming language and its KNIME integration

In addition, we need to install R for the statistical downstream analysis. Choose the directory that matches your operating system, double-click the R installer and follow the instructions. We recommend to use the default settings whenever possible. On macOS you also need to install XQuartz from the same directory.

Afterwards open your R installation. If you use Windows, you will find an "R x64 3.6.X" icon on your desktop. If you use macOS, you will find R in your Applications folder. In R, type the following lines (you might also copy them from the file Rinstall_R_packages.R on the USB stick):

```
install.packages('Rserve',, "http://rforge.net/", type="source")
install.packages("Cairo")

install.packages("devtools")
install.packages("ggplot2")
install.packages("ggfortify")

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install()
BiocManager::install(c("MSstats"))
```

In KNIME, click on **KNIME > Preferences**, select the category **KNIME > R** and set the "Path to R Home" to your installation path. You can use the following settings, if you installed R as described above:

- Windows: C:\Program Files\R\R-3.6.X' (where X is the version you used to install the above libraries)
- macOS: /Library/Frameworks/R.framework/Versions/3.6/Resources

KNIME OpenMS plugin

You are now ready to install the OpenMS nodes.

- In KNIME, click on **Help > Install New Software**

You now have to choose an *update site* to install the OpenMS plugin from. Which *update site* to choose depends on if you received an USB stick in a hands-on Tutorial or if you are doing this Tutorial online.

Online

To install the OpenMS KNIME plugin from the internet, do the following:

1. From the '**Work with:**' drop-down list, select the *update site* 'KNIME Community Extensions (Trusted)' - <https://update.knime.com/community-contributions/trusted/4.6>
2. Now select the following plugin from the "KNIME Community Contributions - Bioinformatics & NGS" category
 - OpenMS
3. Click on **Next** and follow the instructions and after a restart of KNIME the OpenMS nodes will be available in the Node repository under "Community Nodes".

Note: If this does not work for you, report it and while waiting for a reply/fix, try to use an *update site* of an older KNIME version by editing the KNIME version number in the URL or by using our unofficial *update site* at <https://update.knime.com/community-contributions/trusted/4.6>

[//abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/release/latest](http://abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/release/latest)

USB

We included a custom KNIME update site to install the OpenMS KNIME plugins from the USB stick. If you do not have a stick available, please see below.

- 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: OpenMS 3.0.0 UpdateSite

Location: file:/KNIMEUpdateSite/3.0.0/

- After pressing **OK** KNIME will show you all the contents of the added Update Site.

Note: From now on, you can use this repository for plugins in the **Work with:** drop-down list.

- Select the OpenMS nodes in the "Uncategorized" category and click **Next**.
- Follow the instructions and after a restart of KNIME the OpenMS nodes will be available in the Node repository under "Community Nodes".

Online experimental

To install the nightly/experimental version of the OpenMS KNIME plugin from the internet, do the following:

- 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: OpenMS 3.0.0 UpdateSite

Location: <https://abibuilder.cs.uni-tuebingen.de/archive/openms/knime-plugin/updateSite/nightly/>

- After pressing **OK** KNIME will show you all the contents of the added Update Site.

Note: From now on, you can use this repository for plugins in the **Work with:** drop-down list.

- Select the OpenMS nodes in the "Uncategorized" category and click **Next**.
- Follow the instructions and after a restart of KNIME the OpenMS nodes will be available in the Node repository under "Community Nodes".

KNIME concepts

A workflow is a sequence of computational steps applied to a single or multiple input data to process and analyze the data. In KNIME such workflows are implemented graphically by connecting so-called nodes. A node represents a single analysis step in a workflow. Nodes have input and output ports where the data enters 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 of data in KNIME are tables (similar to an excel sheet). Ports that accept tables are marked with a small triangle. For OpenMS nodes, we use a different port type, so called file ports, representing complete files. Those ports are marked by a small blue box. Filled blue boxes represent mandatory inputs and empty blue boxes optional inputs. The same holds for output ports, despite you can deactivate them in the

configuration dialog (double-click on node) under the **OutputTypes** tab. After execution, deactivated ports will be marked with a red cross and downstream nodes will be inactive (not configurable).

A typical OpenMS workflow in KNIME can be divided in two conceptually different parts:

- Nodes for signal and data processing, filtering and data reduction. Here, files are passed between nodes. Execution times of the individual steps are typically longer for these types of nodes as they perform the main computations.
- Downstream statistical analysis and visualization. Here, tables are passed between nodes and mostly internal KNIME nodes or nodes from third-party statistics plugins are used. The transfer from files (produced by OpenMS) and tables usually happens with our provided Exporter and Reader nodes (e.g. MzTabExporter followed by MzTabReader).

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 fails, the node will switch to the red state. Other anomalies and warnings like missing information or empty results will be presented with a yellow exclamation mark above the traffic light. Most nodes will be configured as soon as all input ports are connected. Some nodes need to know about the output of the predecessor and may stay red until the predecessor was executed. If nodes still remain in a red state, probably additional parameters have to be provided in the configuration dialog that can neither be guessed from the data nor filled with sensible defaults. In this case, or if you want to customize the default configuration in general, you can open the configuration dialog of a node with a double-click on the node. For all OpenMS nodes you will see a configuration dialog like the one shown in below figure.

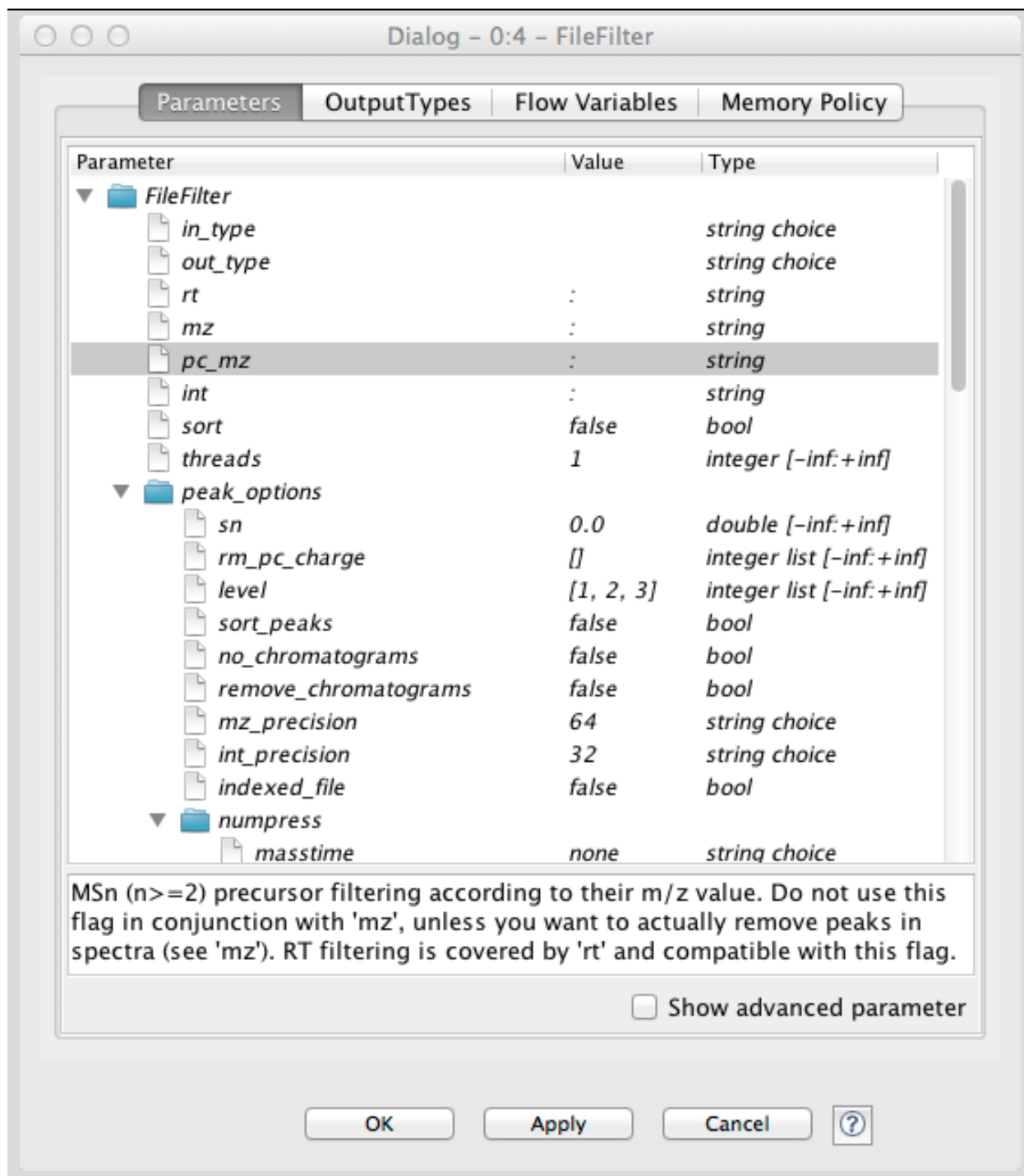


Figure 6: Node configuration dialog of an OpenMS node

Tip: OpenMS distinguishes between normal parameters and advanced parameters. Advanced parameters are by default hidden from the users since they should only rarely be customized. In case you want to have a look at the parameters or need to customize 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. Afterwards the parameters are shown in a light gray color.

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. Please also note the tabs on the top of the configuration dialog. In the case of OpenMS nodes, there will be another tab called OutputTypes. It contains dropdown menus for every output port that let you select the output filetype that you want the node to return (if the tool supports it). For optional output ports you can select Inactive such that the port is crossed out after execution and the associated generation of the file and possible additional computations are not performed. Note that this will deactivate potential downstream nodes connected to this port.

Overview of the graphical user interface

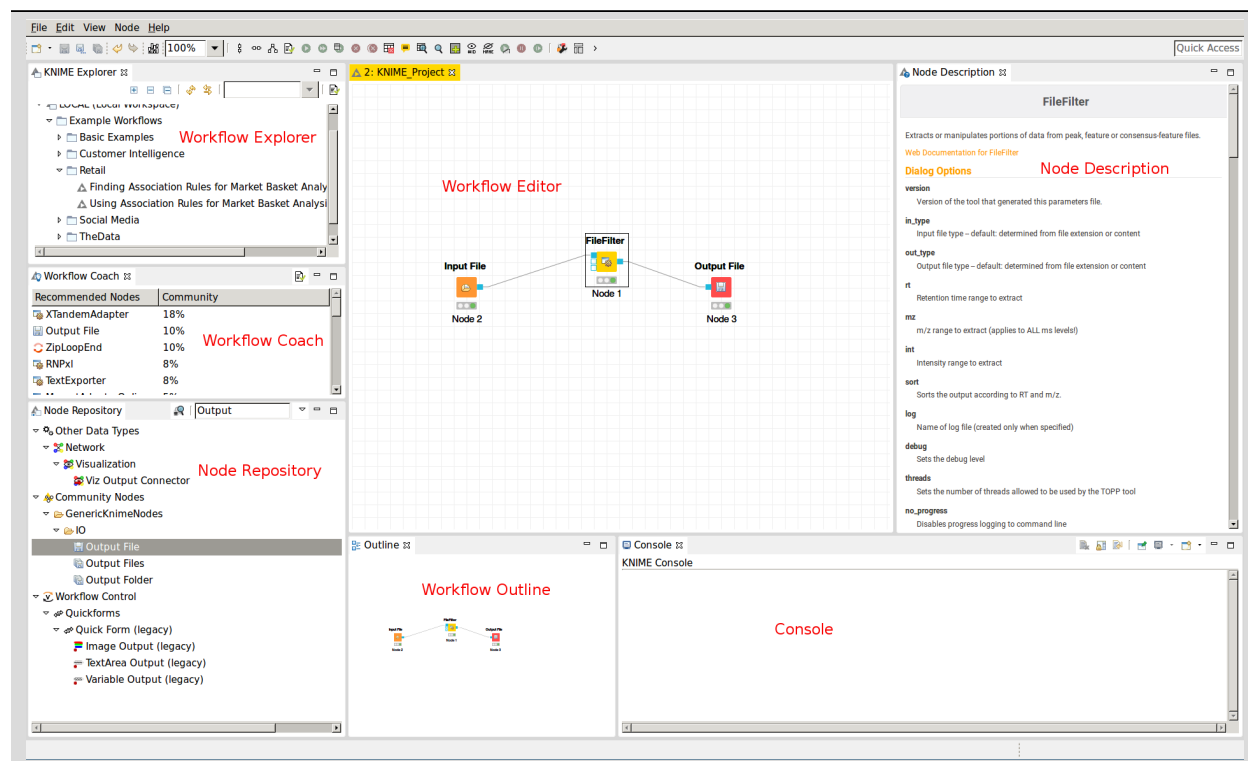


Figure 7: The KNIME workbench

The graphical user interface (GUI) of KNIME consists of different components or so-called panels that are shown in above image. We will briefly introduce the individual panels and their purposes below.

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". For quick creation of a workflow, note that double-clicking on a node in the repository automatically connects it to the selected node in the workbench (connecting all the inputs with as many fitting outputs of the last node). Manually, nodes can be connected by clicking on the output port of one node and dragging the edge until releasing the mouse at the desired input port of the next node. Deletions are possible by selecting nodes and/or edges and pressing DEL or Fn + Backspace depending on your OS and settings. Multiselection happens via dragging rectangles with the mouse or adding elements to the selection by clicking them while holding down Ctrl.

KNIME 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 choosing **New KNIME Workflow** from the appearing context menu. Remember to save your workflow often with the Ctrl + S shortcut.

Workflow Coach (since KNIME 3.2.1)

Shows a list of suggested following nodes, based on the last added/clicked nodes. When you are not sure which node to choose next, you have a reasonable suggestion based on other users behavior there. Connect them to the last node with a double-click.

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 Node > OpenMS Nodes** for managing files (e.g., Input Files or Output Folders) can be found in **Community Nodes > GenericKnimeNode**. 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. You can adjust the zoom level of the explorer by adjusting the percentage in the toolbar at the top of KNIME.

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 a warning 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 and especially their in- and outputs, such that you know what types of data nodes may produce or expect. For OpenMS nodes you will also find a link to the tool page of the online documentation.

Creating workflows

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

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 *knwf* file containing all the information on nodes, their connections, and their parameter configuration.

Those *knwf* 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 on the USB Stick. Additionally, the workflow files can be found on workflow downloads. 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 *knwf* file and after that double-click it in your KNIME Workflow repository to open it.

Duplicating workflows

In this 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, save it, close it and follow the next 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, too.

A minimal workflow

Let us now start with the creation of a 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. This minimal workflow can also be used to check if all requirements are met and that your system is compatible.

- 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 Node > 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.

Tip: 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 below image. FileInfo can produce two different kinds of output files.

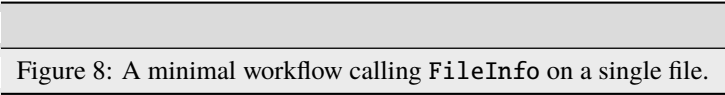


Figure 8: A minimal workflow calling `FileInfo` on a single file.

- 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 `Example_DataIntroductiondatasets005614.mzML` and click **Open**. Afterwards close the dialog by clicking **Ok**.

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 $\uparrow + F7$ (shift key + F7; or the button with multiple green triangles in the KNIME Toolbar) 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.

Workflows are typically constructed to process a large number of files automatically. As a simple example, consider you would like to convert multiple Thermo Raw files into the mzML format. 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.
- 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 **Example_Data > Introduction > datasets > 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 **FileConverter** node, the first output port of the **FileConverter** node to the first input port of the **ZipLoopEnd** node, and the first output port of the **ZipLoopEnd** node to the **Output Folder** node (NOT to the Output File).

The complete workflow is shown in below figure.

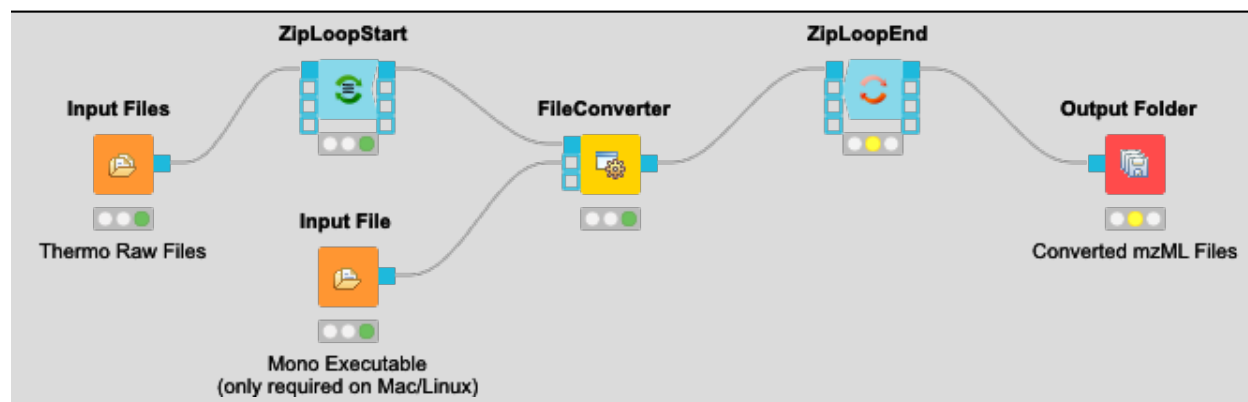


Figure 9: A minimal workflow calling the FileConverter on multiple Thermo Raw files in a loop

Execute the workflow and inspect the output as before.

In case you had trouble to understand what **ZipLoopStart** and **ZipLoopEnd** do, here is a brief explanation:

- The **Input Files** node passes a list of files to the **ZipLoopStart** node.
- The **ZipLoopStart** node takes the files as input, but passes the single files sequentially (that is: one after the other) to the next node.
- The **ZipLoopEnd** collects the single files that arrive at its input port. After all files have been processed, the collected files are passed again as file list to the next node that follows.

Digression: Working with chemical structures

Metabolomics analyses often involve working with chemical structures. Popular cheminformatic toolkits such as RDKit⁷ or CDK⁸ are available as KNIME plugins and allow us to work with chemical structures directly from within KNIME. In particular, we will use KNIME and RDKit to visualize a list of compounds and filter them by predefined sub-structures. Chemical structures are often represented as SMILES (Simplified molecular input line entry specification), a simple and compact way to describe complex chemical structures as text. For example, the chemical structure of L-alanine can be written as the SMILES string C[C@H](N)C(O)=O. As we will discuss later, all OpenMS tools that perform metabolite identification will report SMILES as part of their result, which can then be further processed and visualized using RDKit and KNIME.

⁷ RDKit: Open-source cheminformatics, <http://www.rdkit.org>, [Online; accessed 31-August-2018]. 25

⁸ C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttmann, and E. Willighagen, The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics, Journal of Chemical Information and Computer Sciences 43(2), 493–500 (2003), PMID: 12653513, doi:10.1021/ci025584y. 25

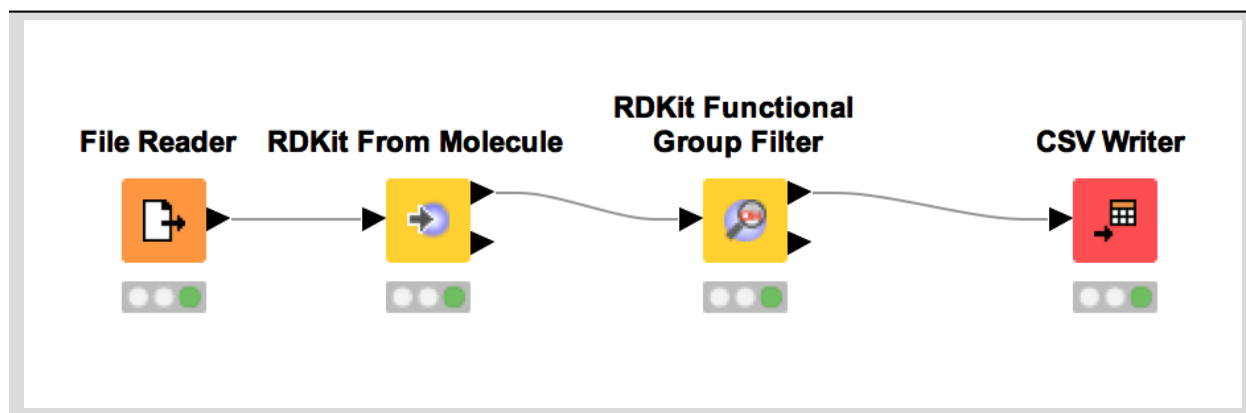


Figure 10: Workflow to visualize a list of SMILES strings and filter them by predefined substructures

Perform the following steps to build the workflow shown in the above figure. You will use this workflow to visualize a list of SMILES strings and filter them by predefined substructures:

- Add the node **File Reader**, open the node configuration dialog and select the file `smiles.csv`. This file has been exported from the Human Metabolome Database (HMDB) and contains the portion of the human metabolome that has been detected and quantified. The file preview on the bottom of the dialog shows that each compound is given by its HMDB accession, compound name, and SMILES string. Click on the column header **SMILES** to change its properties. Change the column type from **string** to **smiles** and close the dialog with **Ok**. Afterwards the **SMILES** column will be visualized as chemical structures instead of text directly within all **KNIME** tables.
- Add the node **RDKit From Molecule** and connect it to the **File Reader**. This node will use the provided SMILES strings to add an additional column that is required by RDKit.
- Add the node **RDKit Functional Group Filter** and open the node configuration dialog. You can use this dialog to filter the compounds by any combination of functional groups. In this case we want to find all compounds that contain at least one aromatic carboxylic acid group. To do this, set this group as active and choose ' C(=O) ' and ' C1= '.
- Connect the first output port (Molecules passing the filter) to a **CSV Writer** node to save the filtered metabolites to a file. Right click **RDKit Functional Group Filter** and select the view 'Molecules passing the filter' to inspect the selected compounds in KNIME. How many compounds pass the chosen filter, see below figure.

The following figure shows resulting list of compounds that contains at least one aromatic carboxylic acid group.

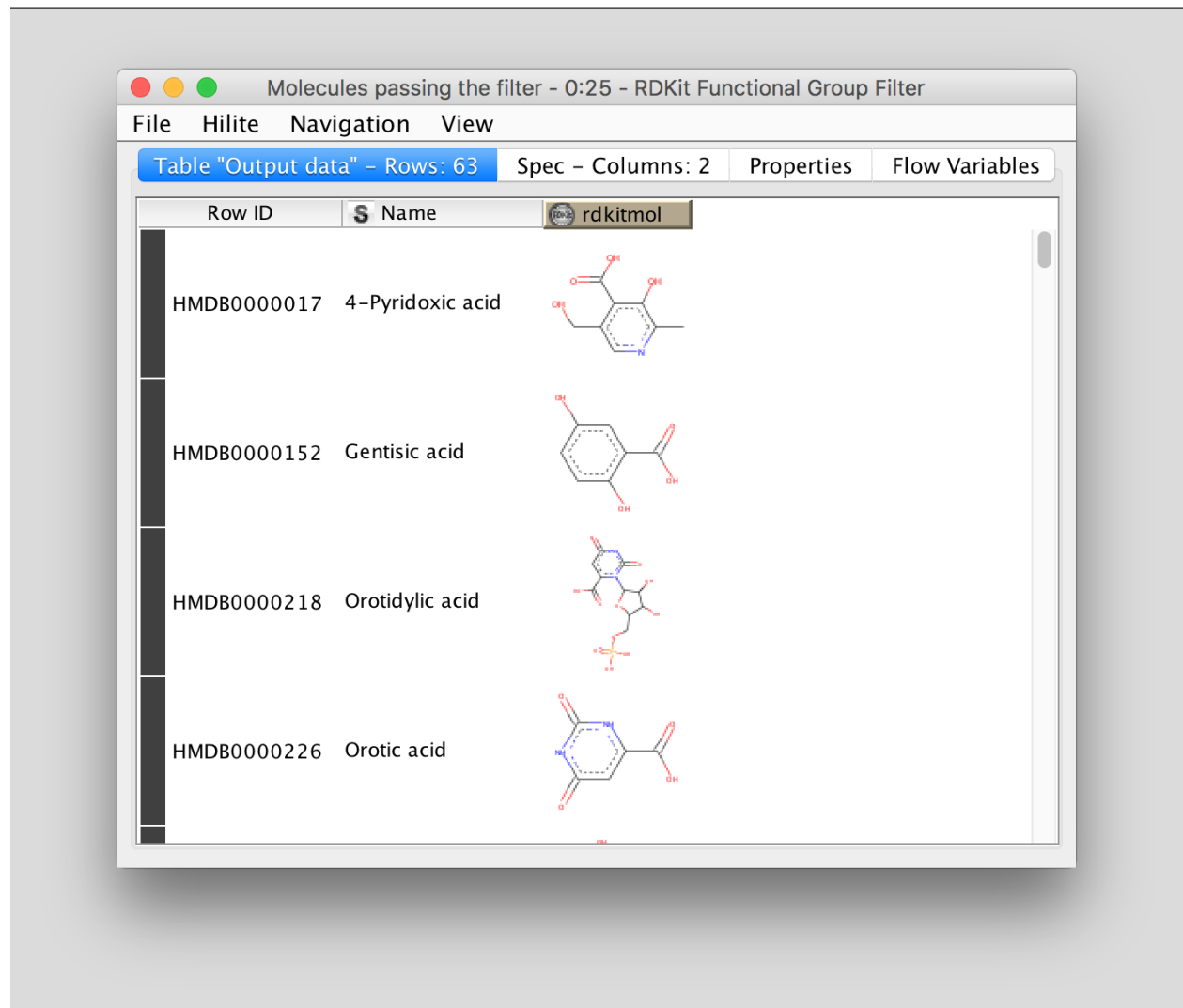


Figure 11: Resulting list of compounds that contains at least one aromatic carboxylic acid group.

Advanced topic: Metanodes

Workflows can get rather complex and may contain dozens or even hundreds of nodes. KNIME provides a simple way to improve handling and clarity of large workflows:

Metanodes allow to bundle several nodes into a single **Metanode**.

Task

Select multiple nodes (e.g. all nodes of the ZipLoop including the start and end node). To select a set of nodes, draw a rectangle around them with the left mouse button or hold Ctrl to add/remove single nodes from the selection.

Tip

There is a **Select Loop** option when you right-click a node in a loop, that does exactly that for you. Then, open the context menu (right-click on a node in the selection) and select **Create Metanode**. Enter a caption for the **Metanode**.

The previously selected nodes are now contained in the **Metanode**. Double-clicking on the **Metanode** will display the contained nodes in a new tab window.

Task

Create the Metanode to let it behave like an encapsulated single node. First select the **Metanode**, open the context menu (right-click) and select **Metanode > Wrap**. The differences between Metanodes and their wrapped counterparts are marginal (and only apply when exposing user inputs and workflow variables). Therefore we suggest to use standard Metanodes to clean up your workflow and cluster common subparts until you actually notice their limits.

Task

Undo the packaging. First select the **(Wrapped) Metanode**, open the context menu (right-click) and select **(Wrapped) Metanode > Expand**.

Advanced topic: R integration

KNIME provides a large number of nodes for a wide range of statistical analysis, machine learning, data processing, and visualization. Still, more recent statistical analysis methods, specialized visualizations or cutting edge algorithms may not be covered in KNIME. In order to expand its capabilities beyond the readily available nodes, external scripting languages can be integrated. In this tutorial, we primarily use scripts of the powerful statistical computing language R. Note that this part is considered advanced and might be difficult to follow if you are not familiar with R. In this case you might skip this part.

R View (Table) allows to seamlessly include R scripts into KNIME. We will demonstrate on a minimal example how such a script is integrated.

Task

First we need some example data in KNIME, which we will generate using the **Data Generator** node. You can keep the default settings and execute the node. The table contains four columns, each containing random coordinates and one column containing a cluster number (Cluster_0 to Cluster_3). Now place a **R View (Table)** node into the workflow and connect the upper output port of the **Data Generator** node to the input of the **R View (Table)** node. Right-click and configure the node. If you get an error message like `Execute failed: R_HOME does not contain a folder with name 'bin'.` or `Execution failed: R Home is invalid.: please change the R settings in the preferences.` To do so open **File > Preferences > KNIME > R** and enter the path to your R installation (the folder that contains the bin directory. (e.g., C:\Program Files\RR-3.4.3)).

If you get an error message like: `"Execute failed: Could not find Rserve package. Please install it in your R installation by running "install.packages('Rserve')"."` You may need to run your R binary as administrator (In windows explorer: right-click "Run as administrator") and enter `install.packages('Rserve')` to install the package.

If R is correctly recognized we can start writing an R script. Consider that we are interested in plotting the first and second coordinates and color them according to their cluster number. In R this can be done in a single line. In the **R view (Table)** text editor, enter the following code:

```
plot(x=knime.in$Universe_0_0, y=knime.in$Universe_0_1, main="Plotting column Universe_0_
↪ 0 vs. Universe_0_1", col=knime.in$"Cluster Membership")
```

Explanation: The table provided as input to the **R View (Table)** node is available as **R data.frame** with name `knime.in`. Columns (also listed on the left side of the R View window) can be accessed in the usual R way by first specifying the

`data.frame` name and then the column name (e.g., `knime.in$Universe_0_0`). `plot` is the plotting function we use to generate the image. We tell it to use the data in column `Universe_0_0` of the dataframe object `knime.in` (denoted as `knime.in$Universe_0_0`) as x-coordinate and the other column `knime.in$Universe_0_1` as y-coordinate in the plot. `main` is simply the main title of the plot and `col` the column that is used to determine the color (in this case it is the `Cluster Membership` column).

Now press the Eval script and Show plot buttons.

Note: Note that we needed to put some extra quotes around `Cluster Membership`. If we omit those, R would interpret the column name only up to the first space (`knime.in$Cluster`) which is not present in the table and leads to an error. Quotes are regularly needed if column names contain spaces, tabs or other special characters like `$` itself.

1.15.3 Label-free quantification of peptides

Introduction

In the following 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 of the last chapter and, step-by-step, build a label-free quantification workflow.

Peptide identification

As a start, we will extend the minimal workflow so that it performs a peptide identification using the OMSSA⁹ 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.

Let's start by replacing the input files in our **Input Files** node by the three mzML files in **Example Data > Labelfree > datasets > lfqxspeikeinxdilutionx1-3.mzML**. This is a reduced toy dataset where each of the three runs contains a constant background of *S. pyogenes* peptides as well as human spike-in peptides in different concentrations.¹⁰

- 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 `Example_DataLabelfreedatabases_py0_sf370_potato_human_target_decoy_with_contaminants.fasta`, and we have a very basic peptide identification workflow.

Note: You might also want to save your new identification workflow under a different name. Have a look at duplicating workflows for information on how to create copies of workflows.

⁹ 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). 30

¹⁰ A. Chawade, M. Sandin, J. Teleman, J. Malmström, and F. Levander, Data Processing Has Major Impact on the Outcome of Quantitative Label-Free LC-MS Analysis, *Journal of Proteome Research* 14(2), 676–687 (2015), PMID: 25407311, arXiv:<http://dx.doi.org/10.1021/pr500665j>, doi:10.1021/pr500665j. 30

- 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 an mzML file in **TOPPView**.
- Here, annotate this spectrum 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**. All identified peptides can be seen using this view. User can also browse the corresponding MS2 spectra.

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

- The search results stored in the idXML file can also be read back into a KNIME table for inspection and subsequent analyses: Add a **TextExporter** node from **Community Nodes > OpenMS > File Handling** to your workflow and connect the output port of your **OMSSAAdapter** (the same port **ZipLoopEnd** is connected to) to its input port. This tool will convert the idXML file to a more human-readable text file which can also be read into a KNIME table using the **IDTextReader** node. Add an **IDTextReader** node (**Community Nodes > OpenMS > Conversion**) after **TextExporter** and execute it. Now you can right click **IDTextReader** and select **ID Table** to browse your peptide identifications.
- From here, you can use all the tools KNIME offers for analyzing the data in this table. As a simple example, add a **Histogram (local)** node (from category **Views - Local (Swing)**) node after **IDTextReader**, double-click it, select **peptide_charge** as Histogram column, hit **OK**, and execute it. Right-clicking and selecting **Interactive View: Histogram view** will open a plot showing the charge state distribution of your identifications.

In the next step, we will tweak the parameters of **OMSSA** to better reflect the instrument's accuracy. 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**. The dataset was recorded using an LTQ Orbitrap XL mass spectrometer, set the precursor mass tolerance to a smaller value, say 5 ppm. Set **precursor_mass_tolerance** to 5 and **precursor_error_units** to ppm.

Note: Whenever you change the configuration of a node, the node as well as all its successors will be reset to the Configured state (all node results are discarded and need to be recalculated by executing the nodes again).

- Set **max_precursor_charge** to 5, in order to also search for peptides with charges up to 5.
- 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 to open the drop down list where you can select the the correct modification.

- A common step in analysis 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. target versus decoy), we have to index the results. To this end, 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 (see below figure).

Tip: 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 “DECOY_”, so we have to set `decoy_string` to `DECOY_` and `decoy_string_position` to `prefix` in the configuration dialog of **PeptideIndexer**.
- Now we can go for the FDR estimation, which the **FalseDiscoveryRate** node will calculate for us (you will find it in **Community Nodes > OpenMS > ID Processing**).
- In order to set the FDR level to 1%, we need an **IDFilter** node from **Community Nodes > OpenMS > ID Processing**. Configuring its parameter `score→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 **IDTextReader** like you did before. 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 Metanode. For this, select all nodes inside the **ZipLoop** (including the **Input File** node) and right-click to select **Collapse into Metanode** and name it **ID**. Metanodes are useful when you construct even larger workflows and want to keep an overview.

The below images shows OMSSA ID pipeline including FDR filtering.

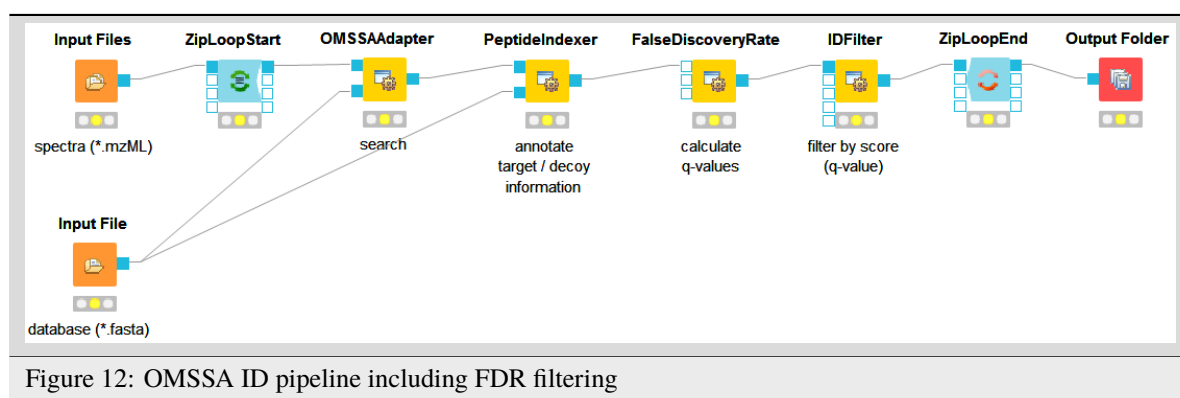


Figure 12: OMSSA ID pipeline including FDR filtering

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. Otherwise, just continue with quantification.

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** and a **XTandemAdapter** **Community Nodes > OpenMS > Identification** node and set its parameters and ports analogously to the **OMSSAAdapter**. In **XTandem**, to get more evenly distributed scores, we decrease the number of candidates a bit by setting the precursor mass tolerance to 5 ppm and the fragment mass tolerance to 0.1 Da.

- 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 > GenericKnlmeNode > 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 for the tutorial files, you have to set **PeptideIndexer's enzyme→specificity** parameter to **none** to accept X!Tandem's non-tryptic identifications as well.

In the end, the ID processing part of the workflow can be collapsed into a Metanode to keep the structure clean (see below figure which shows complete consensus identification workflow).

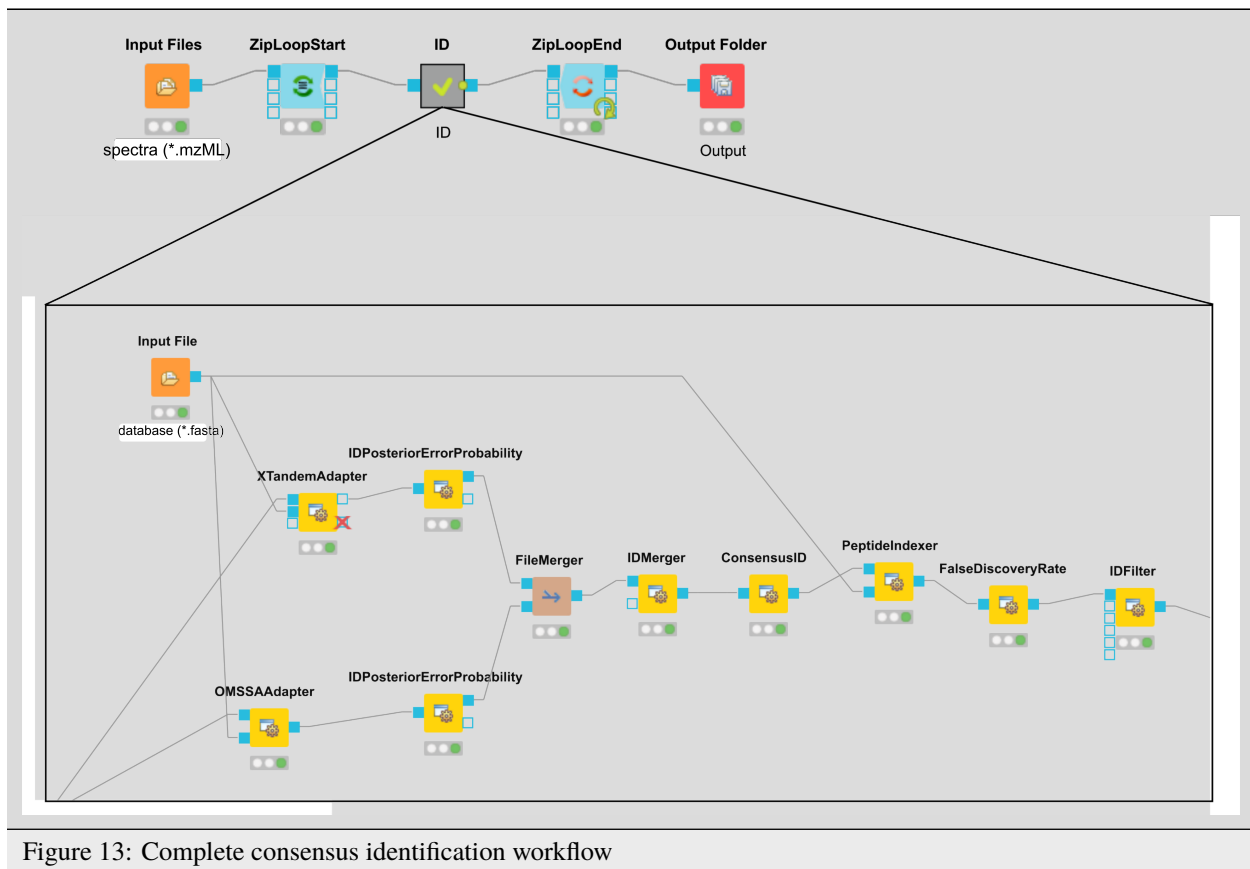


Figure 13: Complete consensus identification workflow

Quantification

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

- Add a **FeatureFinderCentroided** node from **Community Nodes > OpenMS > Quantitation** which gets input from the first output port of the **ZipLoopStart** node. Also, add an **IDMapper** node (from **Community Nodes > OpenMS > ID Processing**) which receives input from the **FeatureFinderCentroided** node (Port 1) and the ID Metanode (or **IDFilter** node (Port 0) if you haven't used the Metanode). The output of the **IDMapper** node is then connected to an in port of 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 overall intensity.
- **FeatureFinderCentroided** produces a featureXML file as output, 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. Open the mzML file of your data to display the raw peak intensities.
- 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. To this end, open the featureXML file in TOPPView by clicking on File Open file and add it to a new layer (Open in New layer). The features are now visualized on top of your raw data. If you zoom in on a small region, you should be able to see the individual boxes around features that have been detected (see Fig. 14). If you hover over the feature centroid (small circle indicating the chromatographic apex of monoisotopic trace) additional information of the feature is displayed.

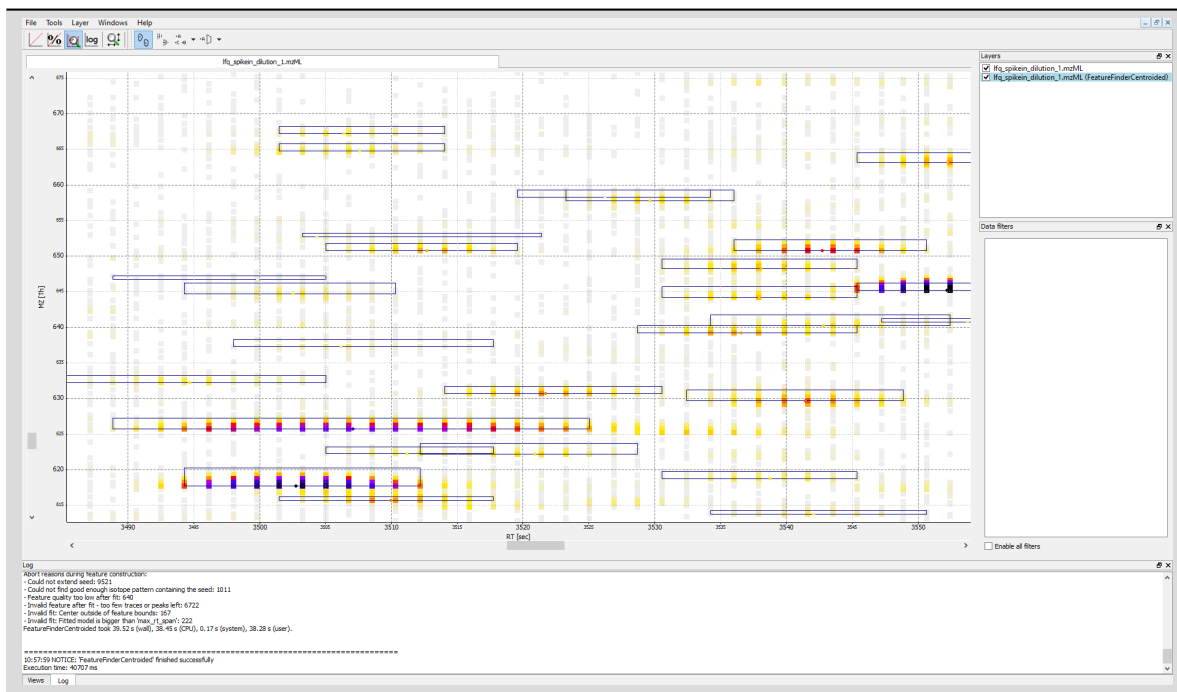


Figure 14: Visualization of detected features (boxes) in TOPPView

Note: The chromatographic RT range of a feature is about 30-60 s and its m/z range around 2.5 m/z in this

dataset. If you have trouble zooming in on a feature, select the full RT range and zoom only into the m/z dimension by holding down Ctrl (cmd on macOS) 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**.

The following image shows the final constructed workflow:

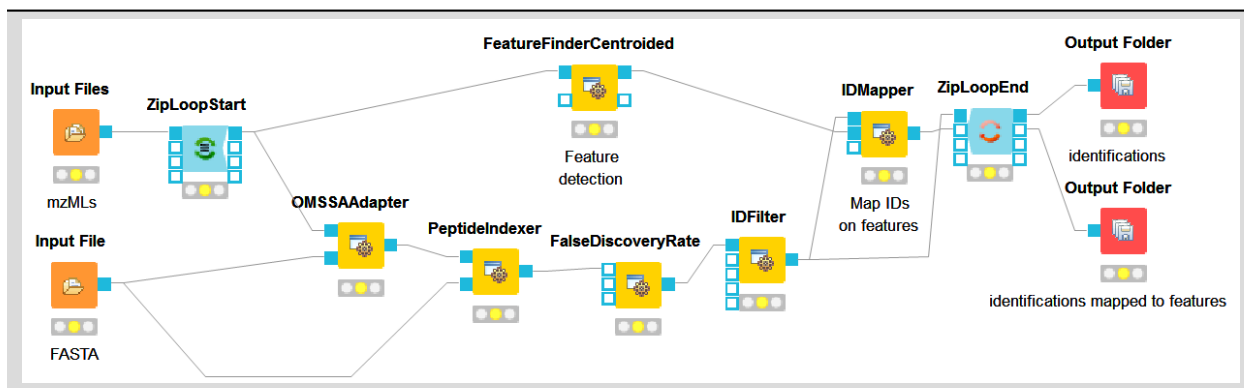


Figure 15: Extended workflow featuring peptide identification and quantification

Combining quantitative information across several label-free experiments

So far, we successfully performed peptide identification as well as quantification on individual LC-MS runs. For differential label-free analyses, however, we need to identify and 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 and extend it a bit further, so that it is able to find and link features across several runs.

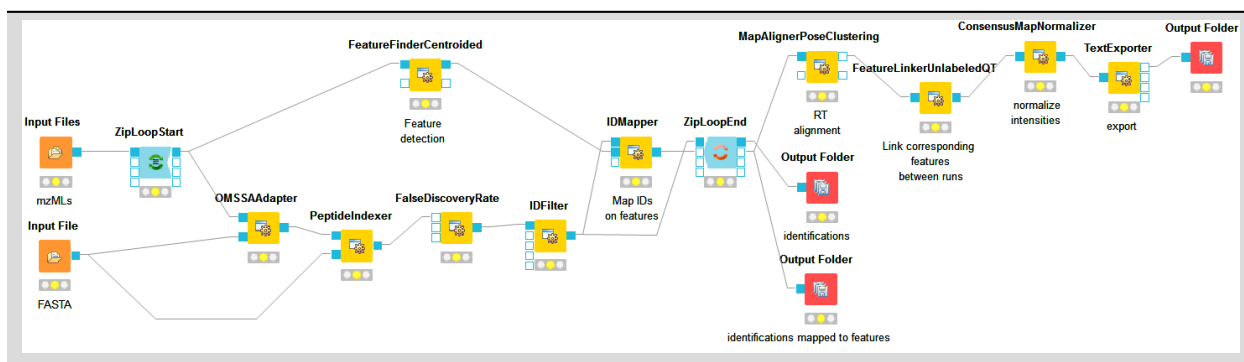


Figure 16: Complete identification and label-free quantification workflow

- 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** node in **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 that **OutputTypes** is set to **featureXML**.

- With the **FeatureLinkerUnlabeledQT** node in **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** node in **Community Node > 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, export the resulting normalized consensusXML file to a csv format using the **TextExporter** node. 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 also be opened with external tools, e.g., Microsoft Excel, for downstream statistical analyses.

Basic data analysis in KNIME

For downstream analysis of the quantification results within the KNIME environment, you can use the **Consensus-TextReader** node in **Community Nodes > OpenMS > Conversion** instead of the **Output Folder** node to convert the output into a KNIME table (indicated by a triangle as output port). After running the node you can view the KNIME table by right-clicking on the **ConsensusTextReader** node and selecting **Consensus Table**. Every row in this table corresponds to a so-called consensus feature, i.e., a peptide signal quantified across several runs. The first couple of columns describe the consensus feature as a whole (average RT and m/z across the maps, charge, etc.). The remaining columns describe the exact positions and intensities of the quantified features separately for all input samples (e.g., **intensity_0** is the intensity of the feature in the first input file). The last 11 columns contain information on peptide identification.

- Now, let’s say we want to plot the log intensity distributions of the human spike-in peptides for all input files. In addition, we will plot the intensity distributions of the background peptides.
- As shown in Fig. 17, add a **Row Splitter** node (**Data Manipulation > Row > Filter**) after the **ConsensusTextReader** node. Double-click it to configure. The human spike-in peptides have accessions starting with “hum”. Thus, set the column to apply the test to: **accessions**, select **pattern matching** as matching criterion, enter **hum** into the corresponding text field, and check the **contains wild cards** box. Press **OK** and execute the node.
- **Row Splitter** produces two output tables: the first one contains all rows from the input table matching the filter criterion, and the second table contains all other rows. You can inspect the tables by right-clicking and selecting **Filtered** and **Filtered Out**. The former table should now only contain peptides with a human accession, whereas the latter should contain all remaining peptides (including unidentified ones).
- Now, since we only want to plot intensities, we can add a **Column Filter** node by going to **Data Manipulation > Column Filter**. Connect its input port to the **Filtered output** port of the **Row Filter** node, and open its configuration dialog. We could either manually select the columns we want to keep, or, more elegantly, select **Wildcard/Regex Selection** and enter **intensity_?** as the pattern. KNIME will interactively show you which columns your pattern applies to while you’re typing.
- Since we want to plot log intensities, we will now compute the log of all intensity values in our table. The easiest way to do this in KNIME is a small piece of R code. Add an **R Snippet** node R after **Column Filter** and double-click to configure. In the R Script text editor, enter the following code:

```

x <- knime.in      # store copy of input table in x

x[x == 0] <- NA    # replace all zeros by NA (= missing value)

x <- log10(x)      # compute log of all values
knime.out <- x     # write result to output table

```

- Now we are ready to plot! Add a **Box Plot (local)** node Views -Swing (local) after the **R Snippet** node, execute it, and open its view. If everything went well, you should see a significant fold change of your human peptide intensities across the three runs.
- To verify that the concentration of background peptides is constant in all three runs, copy and paste the three nodes after **Row Splitter** and connect the duplicated **Column Filter** to the second output port (Filtered Out) of **Row Splitter**, as shown in Fig. 17. Execute and open the view of your second **Box Plot**.

You have now constructed an entire identification and label-free quantification workflow including a simple data analysis using KNIME. The final workflow should like the workflow shown in the following image:

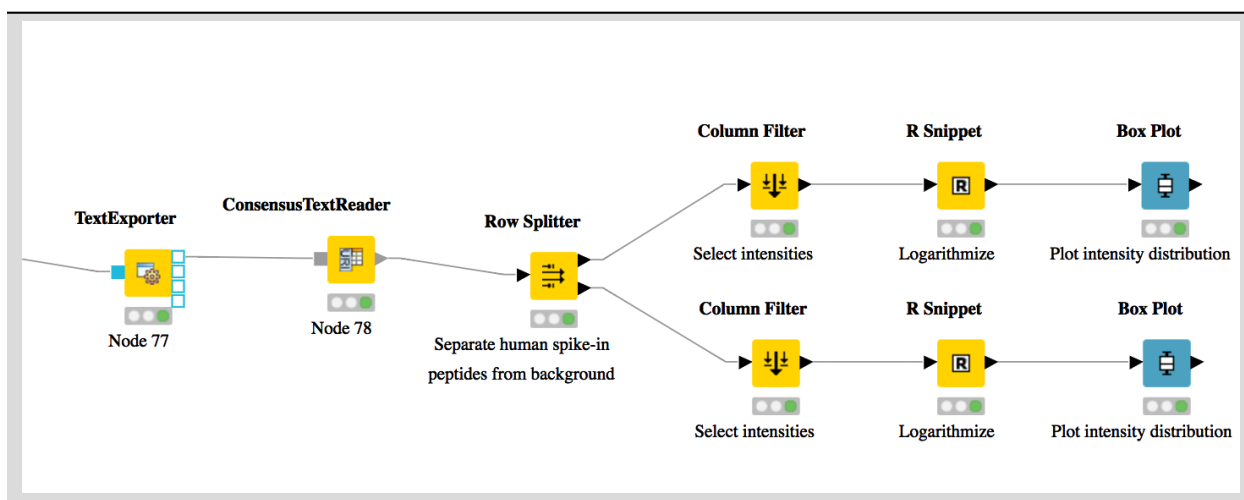


Figure 17: Simple KNIME data analysis example for LFQ

Identification and quantification of the iPRG2015 data with subsequent MSstats analysis

Advanced downstream data analysis of quantitative mass spectrometry-based proteomics data can be performed using MSstats¹¹. This tool can be combined with an OpenMS preprocessing pipeline (e.g. in KNIME). The OpenMS experimental design is used to present the data in an MSstats-conformant way for the analysis. Here, we give an example how to utilize these resources when working with quantitative label-free data. We describe how to use OpenMS and MSstats for the analysis of the ABRF iPRG2015 dataset¹².

Note: Reanalysing the full dataset from scratch would take too long. In the following tutorial, we will focus on just the conversion process and the downstream analysis.

¹¹ A. Chawade, M. Sandin, J. Teleman, J. Malmström, and F. Levander, Data Processing Has Major Impact on the Outcome of Quantitative Label-Free LC-MS Analysis, Journal of Proteome Research 14(2), 676–687 (2015), PMID: 25407311, arXiv:<http://dx.doi.org/10.1021/pr500665j>, doi:10.1021/pr500665j. 30

¹² M. Choi, Z. F. Eren-Dogu, C. Colangelo, J. Cottrell, M. R. Hoopmann, E. A. Kapp, S. Kim, H. Lam, T. A. Neubert, M. Palmblad, B. S. Phinney, S. T. Weintraub, B. MacLean, and O. Vitek, ABRF Proteome Informatics Research Group (iPRG) 2015 Study: Detection of Differentially Abundant Proteins in Label-Free Quantitative LC-MS/MS Experiments, J. Proteome Res. 16(2), 945–957 (2017), doi: 10.1021/acs.jproteome.6b00881. 40

Excursion MSstats

The R package **MSstats** can be used for statistical relative quantification of proteins and peptides in mass spectrometry-based proteomics. Supported are label-free as well as labeled experiments in combination with data-dependent, targeted and data independent acquisition. Inputs can be identified and quantified entities (peptides or proteins) and the output is a list of differentially abundant entities, or summaries of their relative abundance. It depends on accurate feature detection, identification and quantification which can be performed e.g. by an OpenMS workflow. MSstats can be used for data processing & visualization, as well as statistical modeling & inference. Please see [Page 179, 11](#) and the [MSstats](#) website for further information.

Dataset

The iPRG (Proteome Informatics Research Group) dataset from the study in 2015, as described in [Page 179, 12](#), aims at evaluating the effect of statistical analysis software on the accuracy of results on a proteomics label-free quantification experiment. The data is based on four artificial samples with known composition (background: 200 ng *S. cerevisiae*). These were spiked with different quantities of individual digested proteins, whose identifiers were masked for the competition as yeast proteins in the provided database (see Table 1).

Identification and quantification

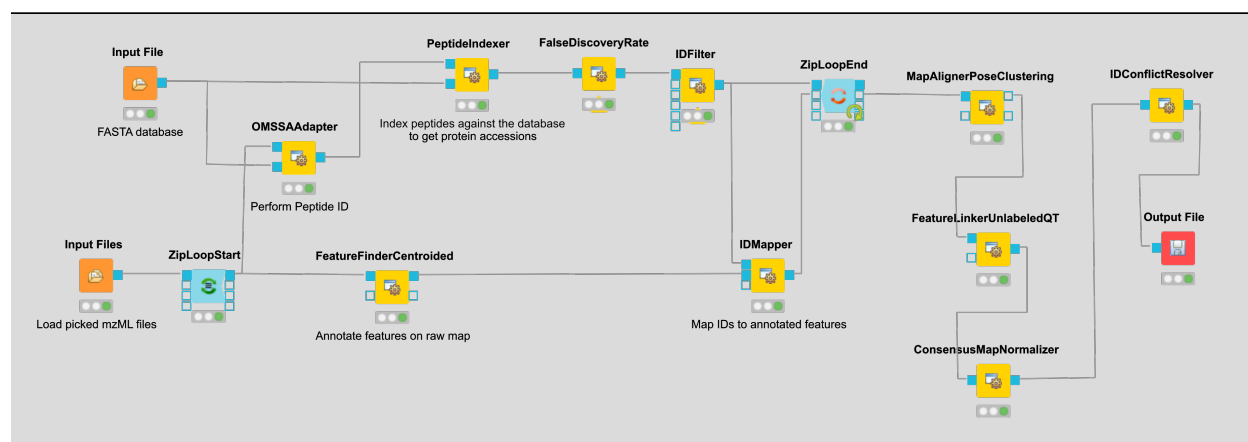


Figure 18: KNIME data analysis of iPRG LFQ data.

The iPRG LFQ workflow (Fig. 18) consists of an identification and a quantification part. The identification is achieved by searching the computationally calculated MS2 spectra from a sequence database (**Input File** node, here with the given database from iPRG: `ExampleData\iPRG2015\database\iPRG2015targetdecoynocontaminants.fasta` against the MS2 from the original data (**Input Files** node with all mzMLs following `ExampleData\iPRG2015\datasets\JD06232014\sample*.mzML` using the **OMSSAAAdapter**.

Note: If you want to reproduce the results at home, you have to download the iPRG data in mzML format and perform peak picking on it or convert and pick the raw data with `msconvert`.

Afterwards, the results are scored using the **FalseDiscoveryRate** node and filtered to obtain only unique peptides (**IDFilter**) since MSstats does not support shared peptides, yet. The quantification is achieved by using the **FeatureFinderCentroided** node, which performs the feature detection on the samples (maps). In the end the quantification

results are combined with the filtered identification results (**IDMapper**). In addition, a linear retention time alignment is performed (**MapAlignerPoseClustering**), followed by the feature linking process (**FeatureLinkerUnlabeledQT**). The **ConsensusMapNormalizers** is used to normalize the intensities via robust regression over a set of maps and the **IDConflictResolver** assures that only one identification (best score) is associated with a feature. The output of this workflow is a consensusXML file, which can now be converted using the **MSstatsConverter** (see Conversion and downstream analysis section).

Experimental design

As mentioned before, the downstream analysis can be performed using MSstats. In this case, an experimental design has to be specified for the OpenMS workflow. The structure of the experimental design used in OpenMS in case of the iPRG dataset is specified in Table 2.

An explanation of the variables can be found in Table 3.

The conditions are highly dependent on the type of experiment and on which kind of analysis you want to perform. For the MSstats analysis the information which sample belongs to which condition and if there are biological replicates are mandatory. This can be specified in further condition columns as explained in Table 3. For a detailed description of the MSstats-specific terminology, see their documentation e.g. in the R vignette.

Conversion and downstream analysis

Conversion of the OpenMS-internal consensusXML format (which is an aggregation of quantified and possibly identified features across several MS-maps) to a table (in MSstats-conformant CSV format) is very easy. First, create a new KNIME workflow. Then, run the **MSstatsConverter** node with a consensusXML and the manually created (e.g. in Excel) experimental design as inputs (loaded via **Input File** nodes). The first input can be found in:

ExampleData\iPRG2015\openmsLFQResults\iPRG\lfq.consensusXML

This file was generated by using the Workflows\openmsLFQiPRG2015.knwf workflow (seen in Fig. 18). The second input is specified in:

ExampleData\iPRG2015\experimental\design.tsv

Adjust the parameters in the config dialog of the converter to match the given experimental design file and to use a simple summing for peptides that elute in multiple features (with the same charge state, i.e. m/z value).

parameter	value
<i>msstats_bioreplicate</i>	MSstats_Bioreplicate
<i>msstats_condition</i>	MSstats_Condition
<i>labeled_reference_peptides</i>	false
<i>retention_time_summarization_method (advanced)</i>	sum

The downstream analysis of the peptide ions with MSstats is performed in several steps. These steps are reflected by several KNIME R nodes, which consume the output of **MSstatsConverter**. The outline of the workflow is shown in Figure 19.

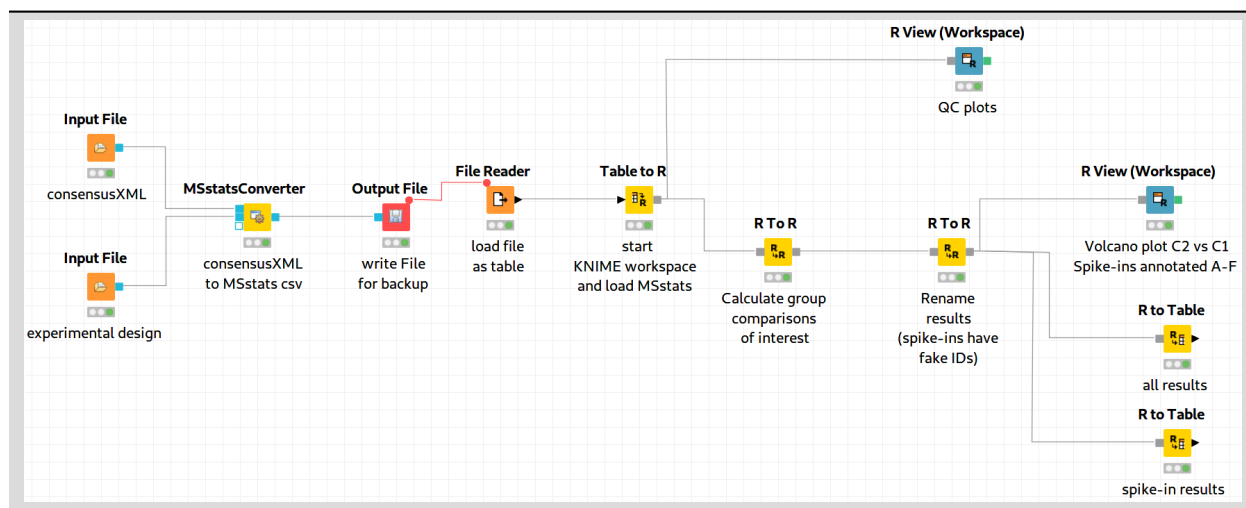


Figure 19: MSstats analysis using KNIME. The individual steps (Preprocessing, Group Comparisons, Result Data Renaming, and Export) are split among several consecutive nodes.

We load the file resulting from **MSstatsConverter** either by saving it with an **Output File** node and reloading it with the **File Reader**. Alternatively, for advanced users, you can use a URI Port to Variable node and use the variable in the File Reader config dialog (V button - located on the right of the **Browse** button) to read from the temporary file.

Preprocessing

The first node (**Table to R**) loads MSstats as well as the data from the previous KNIME node and performs a pre-processing step on the input data. The following inline R script (needs to be pasted into the config dialog of the node):

```
library(MSstats)
data <- knime.in
quant <- OpenMStoMSstatsFormat(data, removeProtein_with1Feature = FALSE)
```

The inline R script allows further preparation of the data produced by **MSstatsConverter** before the actual analysis is performed. In this example, the lines with proteins, which were identified with only one feature, were retained. Alternatively they could be removed. In the same node, most importantly, the following line transforms the data into a format that is understood by MSstats:

```
processed.quant <- dataProcess(quant, censoredInt = 'NA')
```

Here, **dataProcess** is one of the most important functions that the R package provides. The function performs the following steps:

1. Logarithm transformation of the intensities
2. Normalization
3. Feature selection
4. Missing value imputation
5. Run-level summarization

In this example, we just state that missing intensity values are represented by the NA string.

Group Comparison

The goal of the analysis is the determination of differentially-expressed proteins among the different conditions C1-C4. We can specify the comparisons that we want to make in a *comparison* matrix. For this, let's consider the following example:

This matrix has the following properties:

- The number of rows equals the number of comparisons that we want to perform, the number of columns equals the number of conditions (here, column 1 refers to C1, column 2 to C2 and so forth).
- The entries of each row consist of exactly one 1 and one -1, the others must be 0.
- The condition with the entry 1 constitutes the enumerator of the log2 fold-change. The one with entry -1 denotes the denominator. Hence, the first row states that we want calculate:

$$\log_2 \frac{C_2}{C_1}(1.7)$$

We can generate such a matrix in R using the following code snippet in (for example) a new **R to R** node that takes over the R workspace from the previous node with all its variables:

```
comparison1<-matrix(c(-1,1,0,0),nrow=1)
comparison2<-matrix(c(-1,0,1,0),nrow=1)

comparison3<-matrix(c(-1,0,0,1),nrow=1)
comparison4<-matrix(c(0,-1,1,0),nrow=1)

comparison5<-matrix(c(0,-1,0,1),nrow=1)
comparison6<-matrix(c(0,0,-1,1),nrow=1)

comparison <- rbind(comparison1, comparison2, comparison3, comparison4, comparison5,
↪comparison6)
row.names(comparison)<-c("C2-C1", "C3-C1", "C4-C1", "C3-C2", "C4-C2", "C4-C3")
```

Here, we assemble each row in turn, concatenate them at the end, and provide row names for labeling the rows with the respective condition. In MSstats, the group comparison is then performed with the following line:

```
test.MSstats <- groupComparison(contrast.matrix=comparison, data=processed.quant)
```

No more parameters need to be set for performing the comparison.

Result processing

In a next R to R node, the results are being processed. The following code snippet will rename the spiked-in proteins to A,B,C,D,E, and F and remove the names of other proteins, which will be beneficial for the subsequent visualization, as for example performed in Figure 20:

```
test.MSstats.cr <- test.MSstats$ComparisonResult

# Rename spiked ins to A,B,C....
pnames <- c("A", "B", "C", "D", "E", "F")

names(pnames) <- c(
  "sp|P44015|VAC2_YEAST",
  "sp|P55752|ISCB_YEAST",

  "sp|P44374|SFG2_YEAST",
  "sp|P44983|UTR6_YEAST",
  "sp|P44683|PGA4_YEAST",

  "sp|P55249|ZRT4_YEAST"
)

test.MSstats.cr.spikedins <- bind_rows(
  test.MSstats.cr[grep("P44015", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P55752", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P44374", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P44683", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P44983", test.MSstats.cr$Protein),],
  test.MSstats.cr[grep("P55249", test.MSstats.cr$Protein),]
)
# Rename Proteins

test.MSstats.cr.spikedins$Protein <- sapply(test.MSstats.cr.spikedins$Protein, function(x) {pnames[as.character(x)]})

test.MSstats.cr$Protein <- sapply(test.MSstats.cr$Protein, function(x) {

  x <- as.character(x)

  if (x %in% names(pnames)) {

    return(pnames[as.character(x)])
```

(continues on next page)

(continued from previous page)

```

    } else {
      return("")
    }
  })

```

Export

The last four nodes, each connected and making use of the same workspace from the last node, will export the results to a textual representation and volcano plots for further inspection. Firstly, quality control can be performed with the following snippet:

```

qcplot <- dataProcessPlots(processed.quant, type="QCplot",
  ylimDown=0,

  which.Protein = 'allonly',
  width=7, height=7, address=F)

```

The code for this snippet is embedded in the first output node of the workflow. The resulting boxplots show the log2 intensity distribution across the MS runs. The second node is an **R View (Workspace)** node that returns a Volcano plot which displays differentially expressed proteins between conditions C2 vs. C1. The plot is described in more detail in the following Result section. This is how you generate it:

```

groupComparisonPlots(data=test.MSstats.cr, type="VolcanoPlot",

  width=12, height=12,dot.size = 2,ylimUp = 7,

  which.Comparison = "C2-C1",
  address=F)

```

The last two nodes export the MSstats results as a KNIME table for potential further analysis or for writing it to a (e.g. csv) file. Note that you could also write output inside the Rscript if you are familiar with it. Use the following for an **R to Table** node exporting all results:

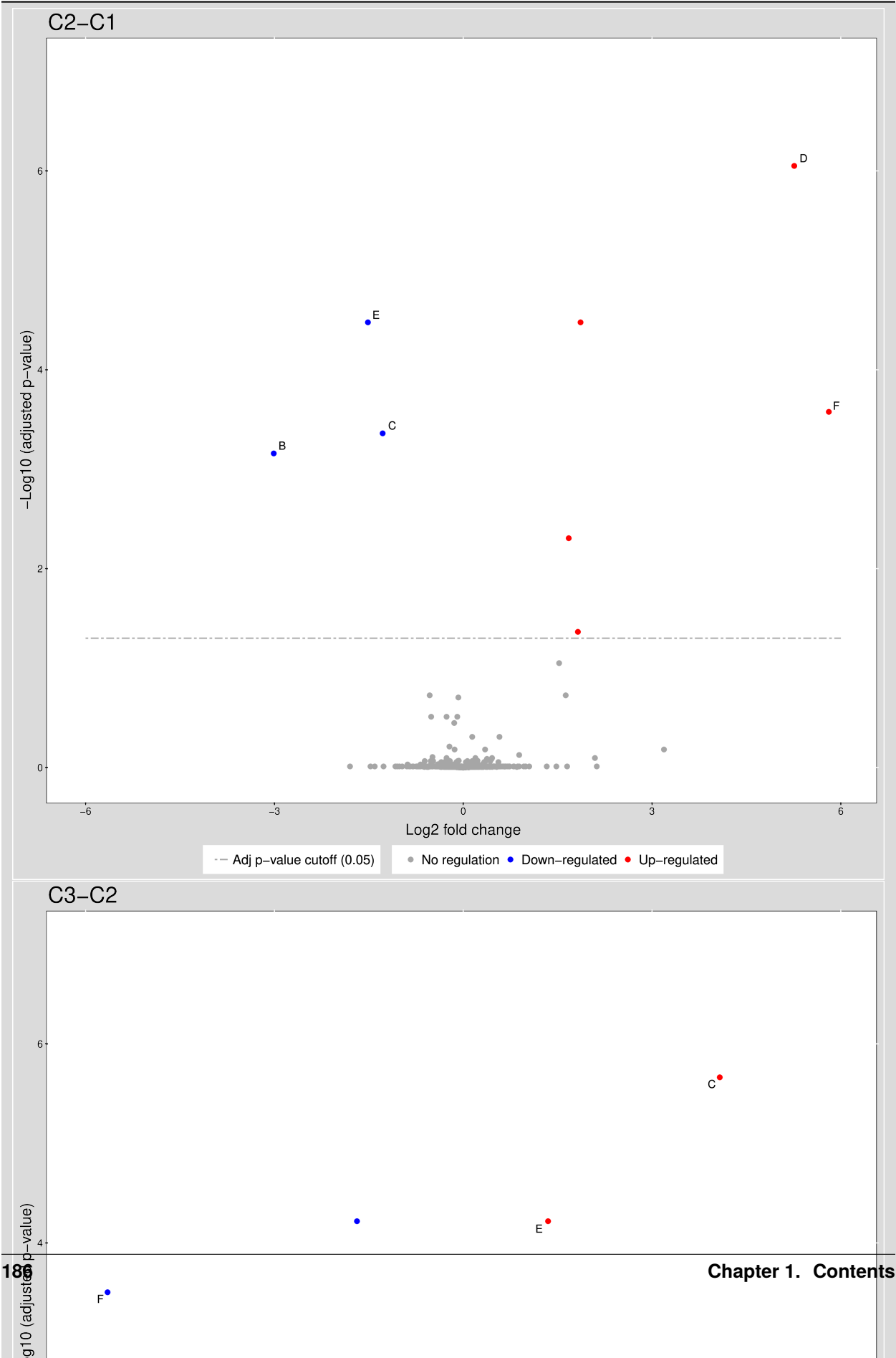
```
knime.out <- test.MSstats.cr
```

And this for an **R to Table** node exporting only results for the spike-ins:

```
knime.out <- test.MSstats.cr.spikedins
```

Result

An excerpt of the main result of the group comparison can be seen in Figure 20.



The Volcano plots show differently expressed spiked-in proteins. In the left plot, which shows the fold-change C2-C1, we can see the proteins D and F (sp|P44983|UTR6_YEAST and sp|P55249|ZRT4_YEAST) are significantly over-expressed in C2, while the proteins B,C, and E (sp|P55752|ISCB_YEAST, sp|P55752|ISCB_YEAST, and sp|P44683|PGA4_YEAST) are under-expressed. In the right plot, which shows the fold-change ratio of C3 vs. C2, we can see the proteins E and C (sp|P44683|PGA4_YEAST and sp|P44374|SFG2_YEAST) over-expressed and the proteins A and F (sp|P44015|VAC2_YEAST and sp|P55249|ZRT4_YEAST) under-expressed. The plots also show further differentially-expressed proteins, which do not belong to the spiked-in proteins.

The full analysis workflow can be found under: WorkflowsMSstatsstatPostProcessingiPRG2015.knwf

1.15.4 Protein inference

In the last chapter, we have successfully quantified peptides in a label-free experiment. As a next step, we will further extend this label-free quantification workflow by protein inference and protein quantification capabilities. This workflow uses some of the more advanced concepts of KNIME, as well as a few more nodes containing R code. For these reasons, you will not have to build it yourself. Instead, we have already prepared and copied this workflow to the USB sticks. Just import Workflowslabelfree_with_protein_quantification.knwf into KNIME via the menu entry **File > Import KNIME workflow > Select file** and double-click the imported workflow in order to open it.

Before you can execute the workflow, you again have to correct the locations of the files in the Input Files nodes (don't forget the one for the FASTA database inside the "ID" meta node). Try and run your workflow by executing all nodes at once.

Extending the LFQ workflow by protein inference and quantification

We have made the following changes compared to the original label-free quantification workflow from the last chapter:

- First, we have added a **ProteinQuantifier** node and connected its input port to the output port of the **ConsensusMapNormalizer** node.
- This already enables protein quantification. **ProteinQuantifier** quantifies peptides by summarizing over all observed charge states and proteins by summarizing over their quantified peptides. It stores two output files, one for the quantified peptides and one for the proteins.
- In this example, we consider only the protein quantification output file, which is written to the first output port of **ProteinQuantifier**.
- Because there is no dedicated node in KNIME to read back the **ProteinQuantifier** output file format into a KNIME table, we have to use a workaround. Here, we have added an additional URI Port to Variable node which converts the name of the output file to a so-called "flow variable" in KNIME. This variable is passed on to the next node **CSV Reader**, where it is used to specify the name of the input file to be read. If you double-click on **CSV Reader**, you will see that the text field, where you usually enter the location of the CSV file to be read, is greyed out. Instead, the flow variable is used to specify the location, as indicated by the small green button with the "v=?" label on the right.
- The table containing the **ProteinQuantifier** results is filtered one more time in order to remove decoy proteins. You can have a look at the final list of quantified protein groups by right-clicking the **Row Filter** and selecting **Filtered**.
- By default, i.e., when the second input port **protein_groups** is not used, **ProteinQuantifier** quantifies proteins using only the unique peptides, which usually results in rather low numbers of quantified proteins.
- In this example, however, we have performed protein inference using Fido and used the resulting protein grouping information to also quantify indistinguishable proteins. In fact, we also used a greedy method in **FidoAdapter** (parameter **greedy_group_resolution**) to uniquely assign the peptides of a group to the most probable protein(s) in the respective group. This boosts the number of quantifications but slightly raises the chances to yield distorted protein quantities.

- As a prerequisite for using **FidoAdapter**, we have added an **IDPosteriorErrorProbability** node within the ID meta node, between the **XTandemAdapter** (note the replacement of OMSSA because of ill-calibrated scores) and **PeptideIndexer**. We have set its parameter `prob_correct` to `true`, so it computes posterior probabilities instead of posterior error probabilities (1 - PEP). These are stored in the resulting idXML file and later on used by the Fido algorithm. Also note that we excluded FDR filtering from the standard meta node. Harsh filtering before inference impacts the calibration of the results. Since we filter peptides before quantification though, no potentially random peptides will be included in the results anyway.
- Next, we have added a third outgoing connection to our ID meta node and connected it to the second input port of **ZipLoopEnd**. Thus, KNIME will wait until all input files have been processed by the loop and then pass on the resulting list of idXML files to the subsequent **IDMerger** node, which merges all identifications from all idXML files into a single idXML file. This is done to get a unique assignment of peptides to proteins over all samples.
- Instead of the meta node **Protein inference with FidoAdapter**, we could have just used a **FidoAdapter** node (**Community Nodes > OpenMS > ID Processing**). However, the meta node contains an additional subworkflow which, besides calling **FidoAdapter**, performs a statistical validation (e.g. (pseudo) receiver operating curves; ROCs) of the protein inference results using some of the more advanced KNIME and R nodes. The meta node also shows how to use **MzTabExporter** and **MzTabReader**.

Statistical validation of protein inference results

In the following section, we will explain the subworkflow contained in the **Protein inference with FidoAdapter** meta node.

Data preparation

For downstream analysis on the protein ID level in KNIME, it is again necessary to convert the idXML-file-format result generated from **FidoAdapter** into a KNIME table.

- We use the **MzTabExporter** to convert the inference results from **FidoAdapter** to a human readable, tab-separated mzTab file. mzTab contains multiple sections, that are all exported by default, if applicable. This file, with its different sections can again be read by the **MzTabReader** that produces one output in KNIME table format (triangle ports) for each section. Some ports might be empty if a section did not exist. Of course, we continue by connecting the downstream nodes with the protein section output (second port).
- Since the protein section contains single proteins as well as protein groups, we filter them for single proteins with the standard **Row Filter**.

ROC curve of protein ID

ROC Curves (Receiver Operating Characteristic curves) are graphical plots that visualize sensitivity (true-positive rate) against fall-out (false positive rate). They are often used to judge the quality of a discrimination method like e.g., peptide or protein identification engines. ROC Curve already provides the functionality of drawing ROC curves for binary classification problems. When configuring this node, select the `opt_global_target_decoy` column as the class (i.e. target outcome) column. We want to find out, how good our inferred protein probability discriminates between them, therefore add `best_search_engine_score[1]` (the inference engine score is treated like a peptide search engine score) to the list of "*Columns containing positive class probabilities*". View the plot by right-clicking and selecting **View: ROC Curves**. A perfect classifier has an area under the curve (AUC) of 1.0 and its curve touches the upper left of the plot. However, in protein or peptide identification, the ground-truth (i.e., which target identifications are true, which are false) is usually not known. Instead, so called pseudoROC Curves are regularly used to plot the number of target proteins against the false discovery rate (FDR) or its protein-centric counterpart, the q-value. The FDR is approximated by using the target-decoy estimate in order to distinguish true IDs from false IDs by separating target IDs from decoy IDs.

Posterior probability and FDR of protein IDs

ROC curves illustrate the discriminative capability of the scores of IDs. In the case of protein identifications, Fido produces the posterior probability of each protein as the output score. However, a perfect score should not only be highly discriminative (distinguishing true from false IDs), it should also be “calibrated” (for probability indicating that all IDs with reported posterior probability scores of 95% should roughly have a 5% probability of being false. This implies that the estimated number of false positives can be computed as the sum of posterior error probabilities ($= 1 - \text{posterior probability}$) in a set, divided by the number of proteins in the set. Thereby a posterior-probability-estimated FDR is computed which can be compared to the actual target-decoy FDR. We can plot calibration curves to help us visualize the quality of the score (when the score is interpreted as a probability as Fido does), by comparing how similar the target-decoy estimated FDR and the posterior probability estimated FDR are. Good results should show a close correspondence between these two measurements, although a non-correspondence does not necessarily indicate wrong results.

The calculation is done by using a simple R script in R snippet. First, the target decoy protein FDR is computed as the proportion of decoy proteins among all significant protein IDs. Then posterior probabilistic-driven FDR is estimated by the average of the posterior error probability of all significant protein IDs. Since FDR is the property for a group of protein IDs, we can also calculate a local property for each protein: the q-value of a certain protein ID is the minimum FDR of any groups of protein IDs that contain this protein ID. We plot the protein ID results versus two different kinds of FDR estimates in R View(Table) (see Fig. 22).

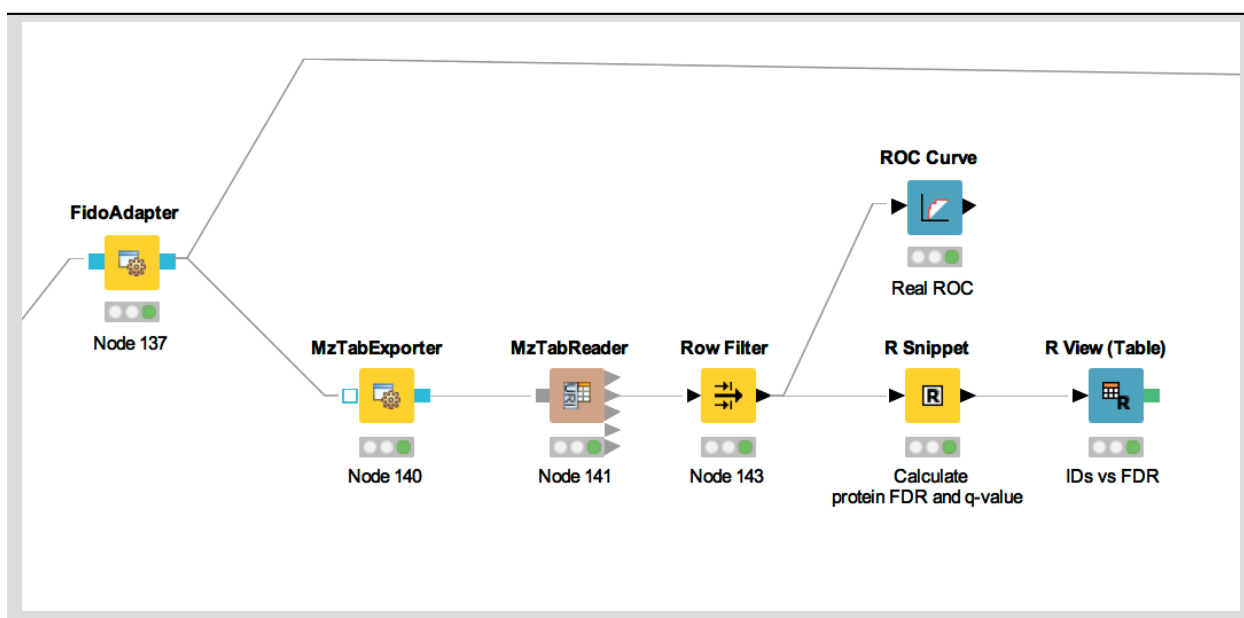


Figure 21: The workflow of statistical analysis of protein inference results

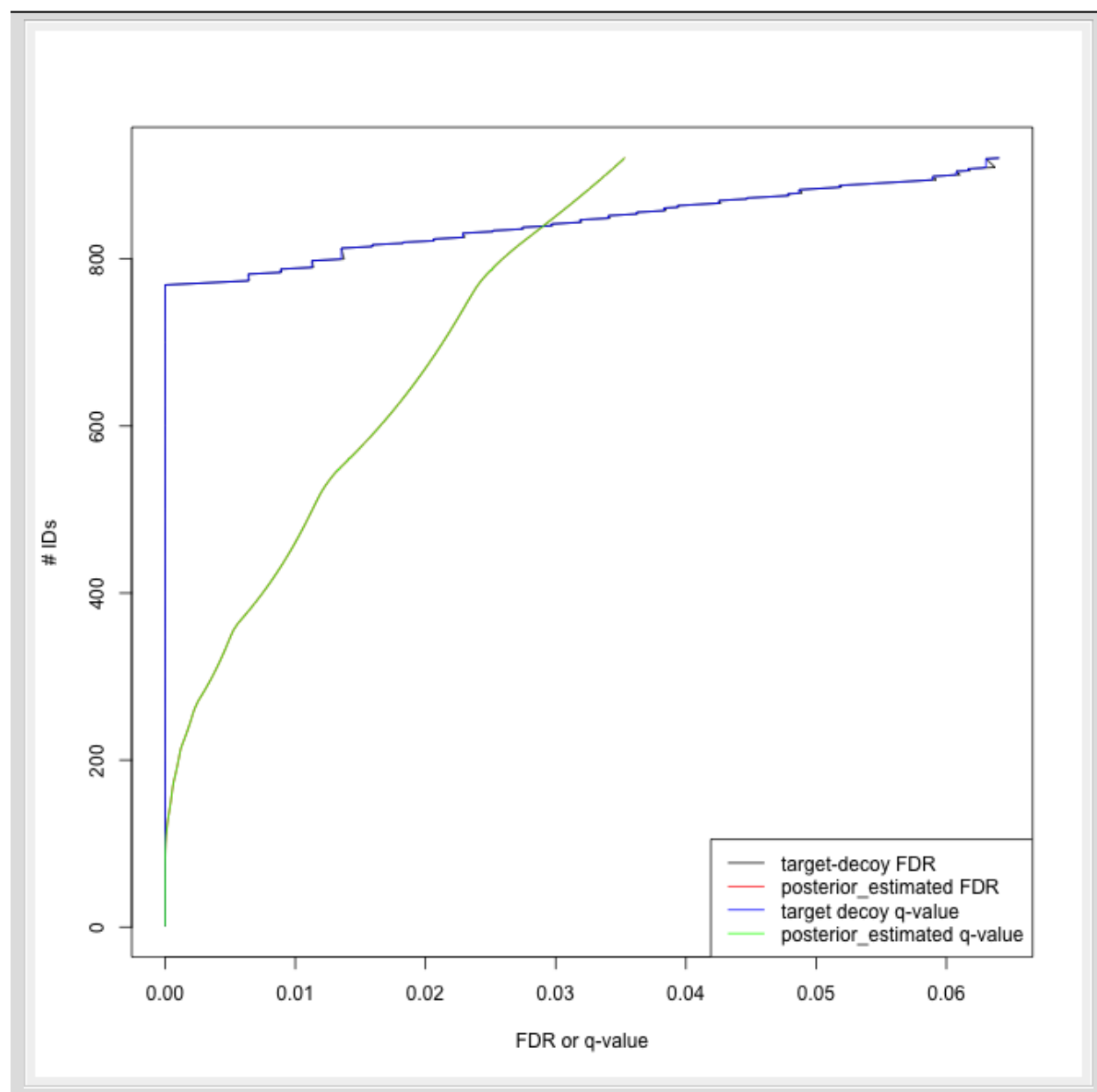


Figure 22: The pseudo-ROC Curve of protein IDs. The accumulated number of protein IDs is plotted on two kinds of scales: target-decoy protein FDR and Fido posterior probability estimated FDR. The largest value of posterior probability estimated FDR is already smaller than 0.04, this is because the posterior probability output from Fido is generally very high

1.15.5 Isobaric analysis

In the last chapters, we identified and quantified peptides in a label-free experiment.

In this section, we would like to introduce a possible workflow for the analysis of isobaric data.

Isobaric analysis workflow

Let's have a look at the workflow (see Fig 23).

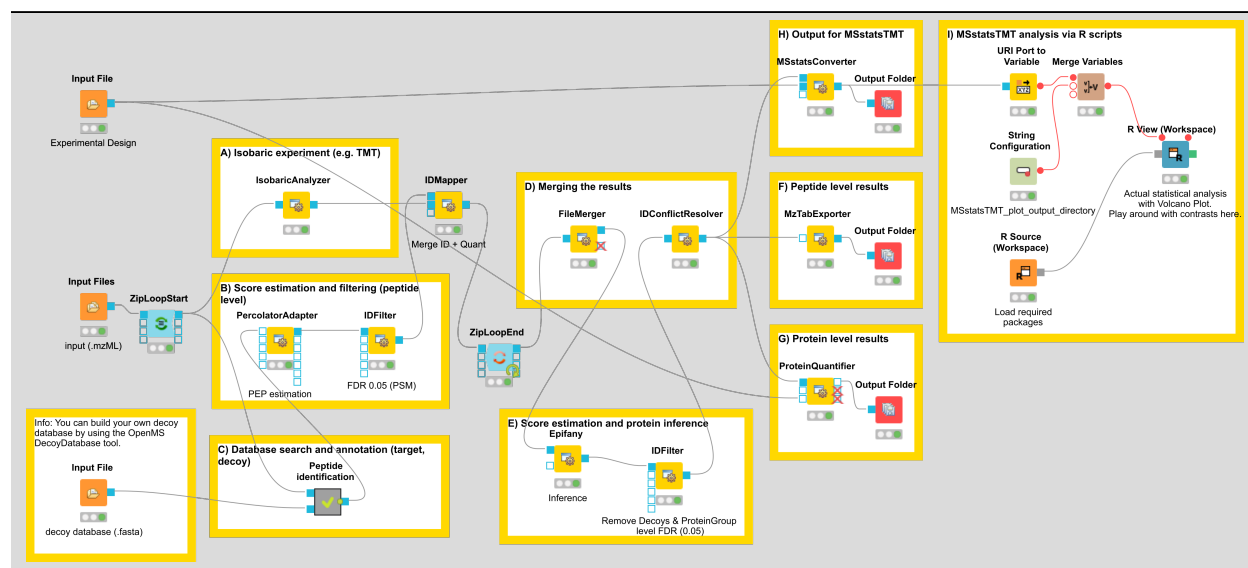


Figure 23: Workflow for the analysis of isobaric data

The full analysis workflow can be found here: [WorkflowsIdentificationquantificationisobaricinferenceepifanyMSstatsTMT](#)

The workflow has four input nodes. The first for the experimental design to allow for MSstatsTMT compatible export (**MSstatsConverter**). The second for the .mzML files with the centroided spectra from the isobaric labeling experiment and the third one for the .fasta database used for identification. The last one allows to specify an output path for the plots generated by the R View, which runs MSstatsTMT (I). The quantification (A) is performed using the **IsobaricAnalyzer**. The tool is able to extract and normalize quantitative information from TMT and iTRAQ data. The values can be assessed from centroided MS2 or MS3 spectra (if available). Isotope correction is performed based on the specified correction matrix (as provided by the manufacturer). The identification © is applied as known from the previous chapters by using database search and a target-decoy database.

To reduce the complexity of the data for later inference the q-value estimation and FDR filtering is performed on PSM level for each file individually (B). Afterwards the identification (PSM) and quantitative information is combined using the **IDMapper**. After the processing of all available files, the intermediate results are aggregated (**FileMerger** - D). All PSM results are used for score estimation and protein inference (**Epifany**) (E). For detailed information about protein inference please see Chapter 4. Then, decoys are removed and the inference results are filtered via a protein group FDR. Peptide level results can be exported via **MzTabExporter** (F), protein level results can be obtained via the **ProteinQuantifier** (G) or the results can be exported (**MSstatsConverter** - H) and further processed with the following R pipeline to allow for downstream processing using MSstatsTMT.

Please import the workflow from WorkflowsIdentificationquantificationisobaricinferenceepifanyMSstatsTMT into KNIME via the menu entry **File > Import KNIME workflow > Select file** and double-click the imported workflow in order to open it. Before you can execute the workflow, you have to correct the locations of the files in the **Input Files**

nodes (don't forget the one for the FASTA database inside the "ID" meta node). Try and run your workflow by executing all nodes at once.

Excursion MSstatsTMT

The R package MSstatsTMT can be used for protein significance analysis in shotgun mass spectrometry-based proteomic experiments with tandem mass tag (TMT) labeling. MSstatsTMT provides functionality for two types of analysis & their visualization: Protein summarization based on peptide quantification and Model-based group comparison to detect significant changes in abundance. It depends on accurate feature detection, identification and quantification which can be performed e.g. by an OpenMS workflow.

In general, MSstatsTMT can be used for data processing & visualization, as well as statistical modeling. Please see¹³ and the [MSstats](#) website for further information.

There is also an [online lecture](#) and tutorial for MSstatsTMT from the May Institute Workshop 2020.

Dataset and experimental design

We are using the MSV000084264 ground truth dataset, which consists of TMT10plex controlled mixes of different concentrated UPS1 peptides spiked into SILAC HeLa peptides measured in a dilution series <https://www.omicsdi.org/dataset/massive/MSV000084264>. Figure 24 shows the experimental design. In this experiment, 5 different TMT10plex mixtures – different labeling strategies – were analysed. These were measured in triplicates represented by the 15 MS runs (3 runs each). The example data, database and experimental design to run the workflow can be found [here](#).

TMT10plex reagent		126	127N	127C	128N	128C	129N	129C	130N	130C	131
Mixture 1	Run 1	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 2	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 3	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
Mixture 2	Run 4	Ref	0.5	1	0.667	0.125	1	0.667	0.125	0.5	Ref
	Run 5	Ref	0.5	1	0.667	0.125	1	0.667	0.125	0.5	Ref
	Run 6	Ref	0.5	1	0.667	0.125	1	0.667	0.125	0.5	Ref
Mixture 3	Run 7	Ref	0.125	0.667	1	0.5	0.5	0.125	0.667	1	Ref
	Run 8	Ref	0.125	0.667	1	0.5	0.5	0.125	0.667	1	Ref
	Run 9	Ref	0.125	0.667	1	0.5	0.5	0.125	0.667	1	Ref
Mixture 4	Run 10	Ref	1	0.5	0.125	0.667	0.667	1	0.5	0.125	Ref
	Run 11	Ref	1	0.5	0.125	0.667	0.667	1	0.5	0.125	Ref
	Run 12	Ref	1	0.5	0.125	0.667	0.667	1	0.5	0.125	Ref
Mixture 5	Run 13	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 14	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref
	Run 15	Ref	0.667	0.125	0.5	1	0.125	0.5	1	0.667	Ref

Figure 24: Experimental Design

The experimental design in table format allows for MSstatsTMT compatible export. The design is represented by two tables. The first one represents the overall structure of the experiment in terms of samples, fractions, labels and

¹³ T. Huang, M. Choi, S. Hao, and O. Vitek, MSstatsTMT: Protein Significance Analysis in shotgun mass spectrometry-based proteomic experiments with tandem mass tag (TMT) labeling., (2020), doi:10.18129/B9.bioc.MSstatsTMT. 55

fraction groups. The second one 5 adds to the first by specifying specific conditions, biological replicates as well as mixtures and label for each channel. For additional information about the experimental design please see Table 3.

After running the workflow, the **MSstatsConverter** will convert the OpenMS output in addition with the experimental design to a file (.csv) which can be processed by using MSstatsTMT.

MSstatsTMT analysis

Here, we depict the analysis by MSstatsTMT using a segment of the isobaric analysis workflow (Fig. 25). The segment is available as WorkflowsMSstatsTMT.knwf.

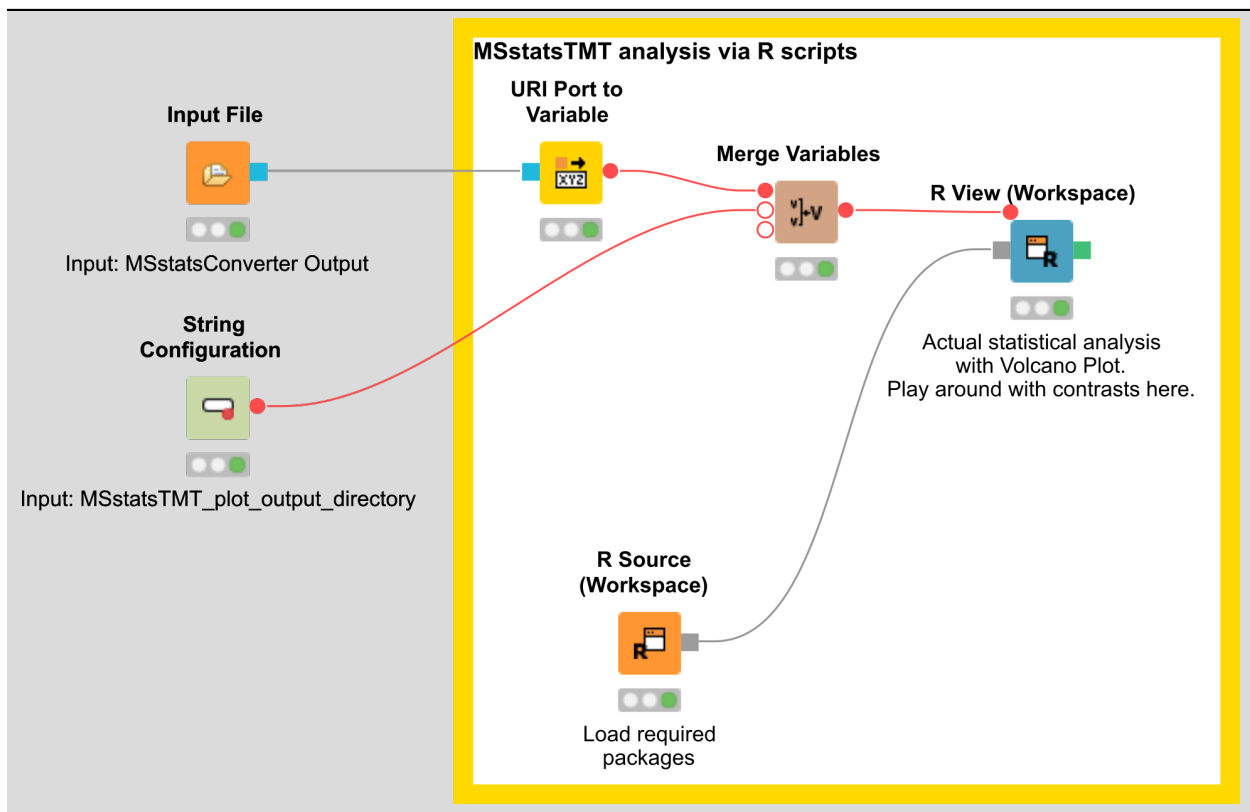


Figure 25: MSstatsTMT workflow segment

There are two input nodes, the first one takes the result (.csv) from the **MSstatsConverter** and the second a path to the directory where the plots generated by MSstatsTMT should be saved. The **R** source node loads the required packages, such as dplyr for data wrangling, MSstatsTMT for analysis and MSstats for plotting. The inputs are further processed in the **R View** node.

Here, the data of the **Input File** is loaded into **R** using the flow variable ["URI-0"]:

```
file <- substr(knime.flow.in[["URI-0"]], 6, nchar(knime.flow.in[["URI-0"]]))

MSstatsConverter_OpenMS_out <- read.csv(file)
data <- MSstatsConverter_OpenMS_out
```

The `OpenMStoMSstatsTMTFormat` function preprocesses the OpenMS report and converts it into the required input format for `MSstatsTMT`, by filtering based on unique peptides and measurements in each MS run.

```
processed.data <- OpenMStoMSstatsTMTFormat(data)
```

Afterwards different normalization steps are performed (global, protein, runs) as well as data imputation by using the `msstats` method. In addition peptide level data is summarized to protein level data.

```
quant.data <- proteinSummarization(processed.data,
                                   method="msstats",

                                   global_norm=TRUE,
                                   reference_norm=TRUE,

                                   MBimpute = TRUE,
                                   maxQuantileforCensored = NULL,

                                   remove_norm_channel = TRUE,
                                   remove_empty_channel = TRUE)
```

There a lot of different possibilities to configure this method please have a look at the `MSstatsTMT` package for [additional detailed information](#).

The next step is the comparions of the different conditions, here either a pairwise comparision can be performed or a confusion matrix can be created. The goal is to detect and compare the UPS peptides spiked in at different concentrations.

```
# prepare contrast matrix
unique(quant.data$Condition)

comparison<-matrix(c(-1,0,0,1,
                     0,-1,0,1,
                     0,0,-1,1,
                     0,1,-1,0,
                     1,-1,0,0), nrow=5, byrow = T)

# Set the names of each row
row.names(comparison)<- contrasts <- c("1-0125",
                                       "1-05",
                                       "1-0667",
                                       "05-0667",
                                       "0125-05")

# Set the column names
colnames(comparison)<- c("0.125", "0.5", "0.667", "1")
```

The constructed confusion matrix is used in the `groupComparisonTMT` function to test for significant changes in protein abundance across conditions based on a family of linear mixed-effects models in TMT experiments.

```
data.res <- groupComparisonTMT(data = quant.data,
                              contrast.matrix = comparison,

                              moderated = TRUE, # do moderated t test

                              adj.method = "BH") # multiple comparison adjustment
data.res <- data.res %>% filter(!is.na(Protein))
```

In the next step the comparison can be plotted using the `groupComparisonPlots` function by `MSstats`.

```
library(MSstats)
groupComparisonPlots(data=data.res.mod, type="VolcanoPlot", address=F, which.Comparison_
↳ = "0125-05", sig = 0.05)
```

Here, we have an example output of the **R View**, which depicts the significant regulated UPS proteins in the comparison of 125 to 05 (Fig. 26).



All plots are saved to the in the beginning specified output directory in addition.

Note

The isobaric analysis does not always have to be performed on protein level, for example for phosphoproteomics studies one is usually interested on the peptide level - in addition inference on peptides with post-translational modification is not straight forward. Here, we present an additional workflow on peptide level, which can potentially be adapted and used for such cases. Please see [WorkflowsIdentificationquantificationisobaricMSstatsTMT](#)

1.15.6 Label-free quantification of metabolites

Introduction

Quantification and identification of chemical compounds are basic tasks in metabolomic studies. In this tutorial session we construct a UPLC-MS based, label-free quantification and identification workflow. Following quantification and identification we then perform statistical downstream analysis to detect quantification 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. The goal of this tutorial is to detect and quantify these differential spike-in compounds against the complex background.

Basics of non-targeted metabolomics data analysis

For the metabolite quantification we choose an approach similar to the one used for peptides, but this time based on the OpenMS `FeatureFinderMetabo` method. This feature finder again collects peak picked data into individual mass traces. The reason why we need a different feature finder for metabolites lies in the step after trace detection: the aggregation of isotopic traces belonging to the same compound ion into the same feature. Compared to peptides with their average model, small molecules have very different isotopic distributions. To group small molecule mass traces correctly, an aggregation model tailored to small molecules is thus needed.

- Create a new workflow called for instance "Metabolomics".
- Add an **Input File** node and configure it with one mzML file from the `Example_DataMetabolomicsdatasets`.
- Add a **FeatureFinderMetabo** node (from **Community Nodes > OpenMS > Quantitation**) and connect the first output port of the **Input File** to the **FeatureFinderMetabo**.
- For an optimal result adjust the following settings. Please note that some of these are advanced parameters.
- Connect a **Output Folder** to the output of the **FeatureFinderMetabo** (see Fig. 27).

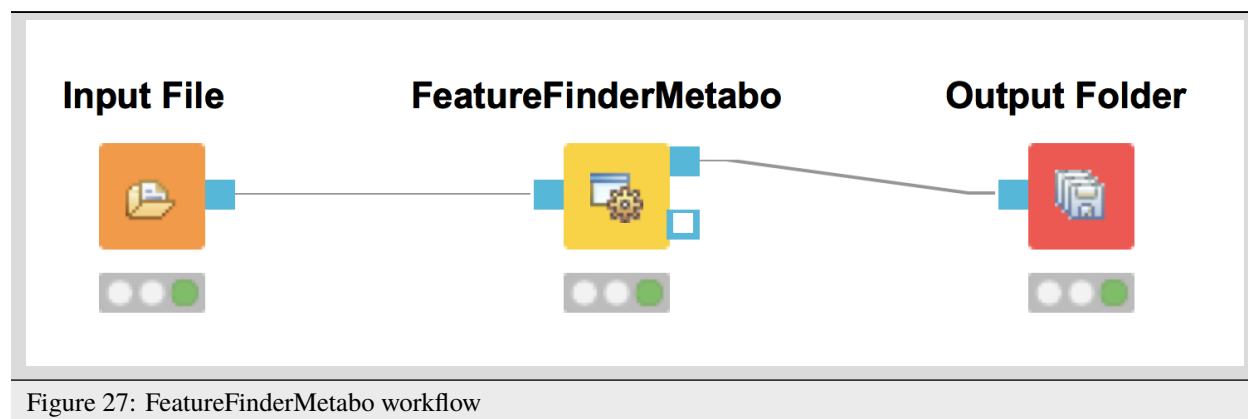


Figure 27: FeatureFinderMetabo workflow

In the following advanced parameters will be highlighted. These parameter can be altered if the `Show advanced parameter` field in the specific tool is activated.

parameter	value
<i>algorithm→common→chrom_fwhm</i>	8.0
<i>algorithm→mtd→trace_termination_criterion</i>	sample_rate
<i>algorithm→mtd→min_trace_length</i>	3.0
<i>algorithm→mtd→max_trace_length</i>	600.0
<i>algorithm→epd→width_filtering</i>	off
<i>algorithm→ffm→report_convex_hulls</i>	true

The parameters change the behavior of **FeatureFinderMetabo** as follows:

- **chrom_fwhm**: The expected chromatographic peak width in seconds.
- **trace_termination_criterion**: In the first stage **FeatureFinderMetabo** assembles mass traces with a pre-defined mass accuracy. If this parameter is set to 'outlier', the extension of a mass trace is stopped after a predefined number of consecutive outliers is found. If this parameter is set to 'sample_rate', the extension of a mass trace is stopped once the ratio of collected peaks versus visited spectra falls below the ratio given by `min_sample_rate`.
- **min_trace_length**: Minimal length of a mass trace in seconds. Choose a small value, if you want to identify low-intensity compounds.
- **max_trace_length**: Maximal length of a mass trace in seconds. Set this parameter to -1 to disable the filtering by maximal length.
- **width_filtering**: **FeatureFinderMetabo** can remove features with unlikely peak widths from the results. If activated it will use the interval provided by the parameters `min_fwhm` and `max_fwhm`.
- **report_convex_hulls**: If set to true, convex hulls including mass traces will be reported for all identified features. This increases the output size considerably.

The output file `.featureXML` can be visualized with TOPPView on top of the used `.mzML` file - in a so called layer - to look at the identified features.

First start TOPPView and open the example `.mzML` file (see Fig. 28). Afterwards open the `.featureXML` output as new layer (see Fig. 29). The overlay is depicted in Figure 30. The zoom of the `.mzML` - `.featureXML` overlay shows the individual mass traces and the assembly of those in a feature (see Fig. 31).

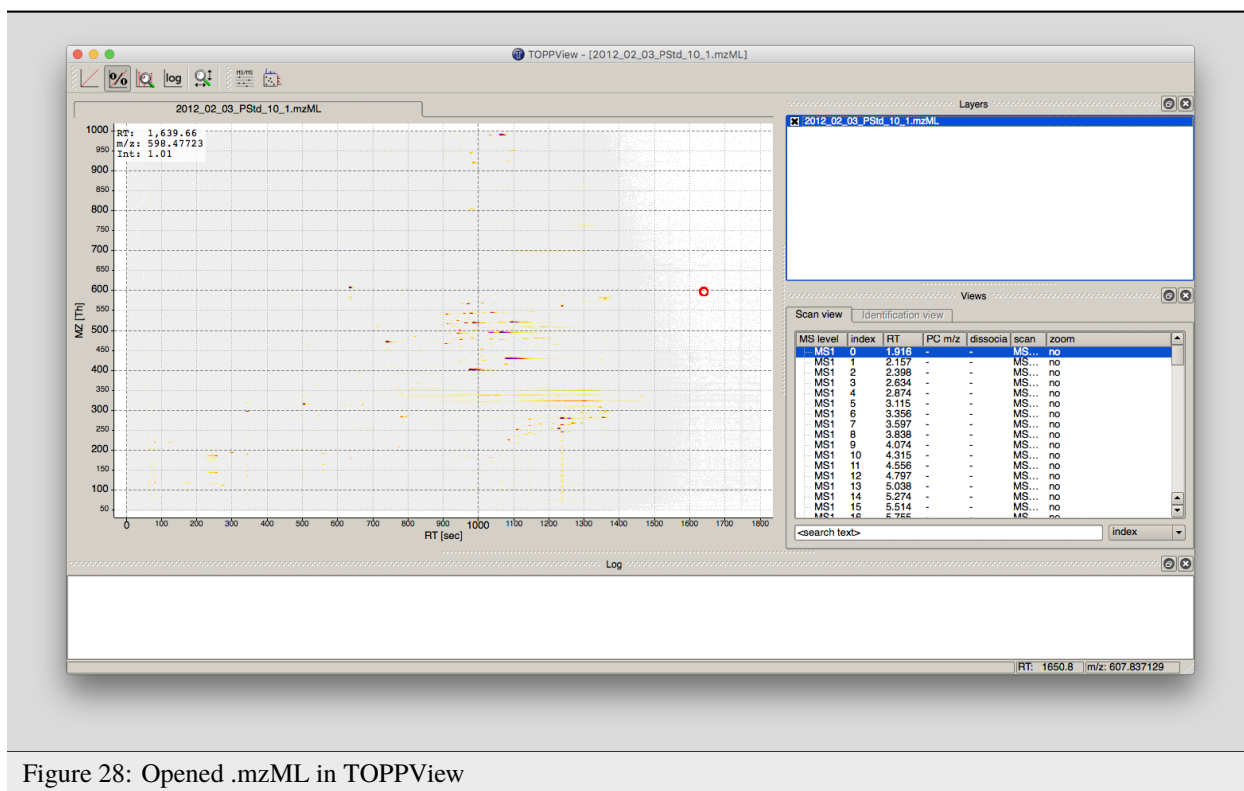


Figure 28: Opened .mzML in TOPPView

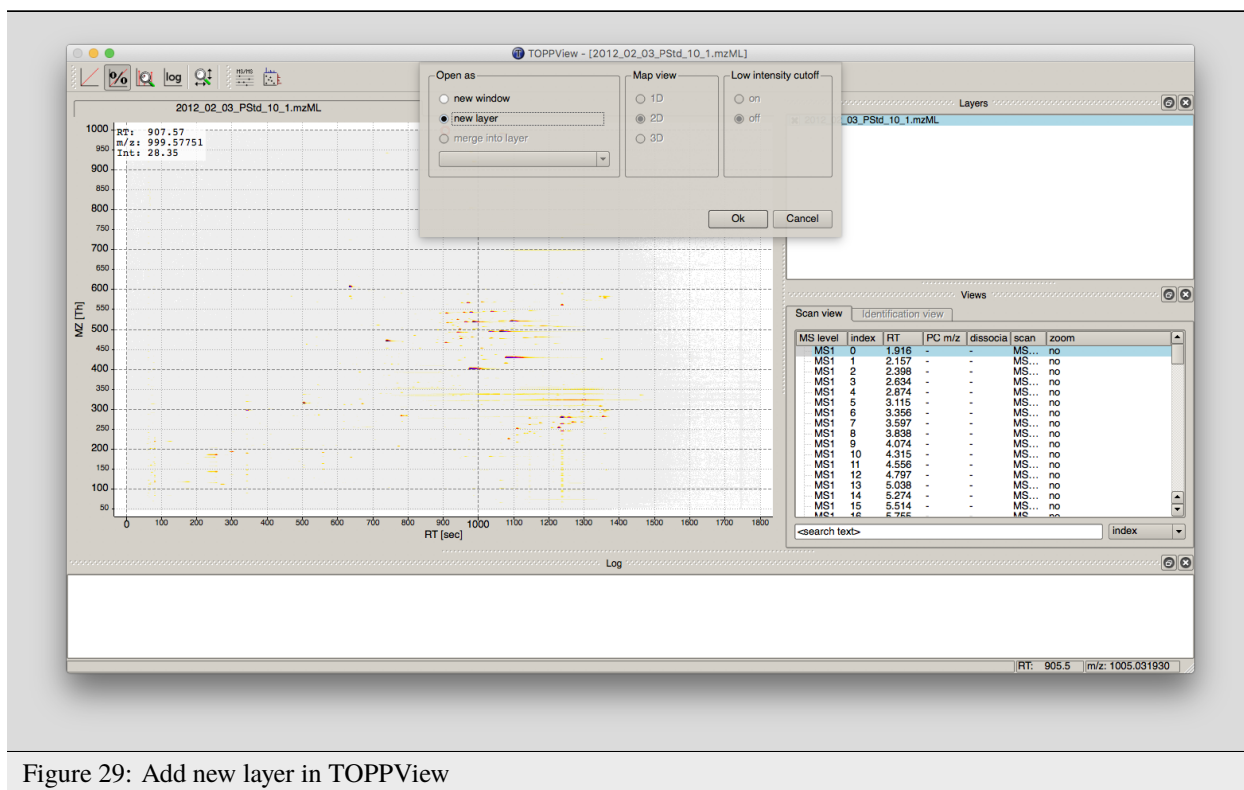


Figure 29: Add new layer in TOPPView

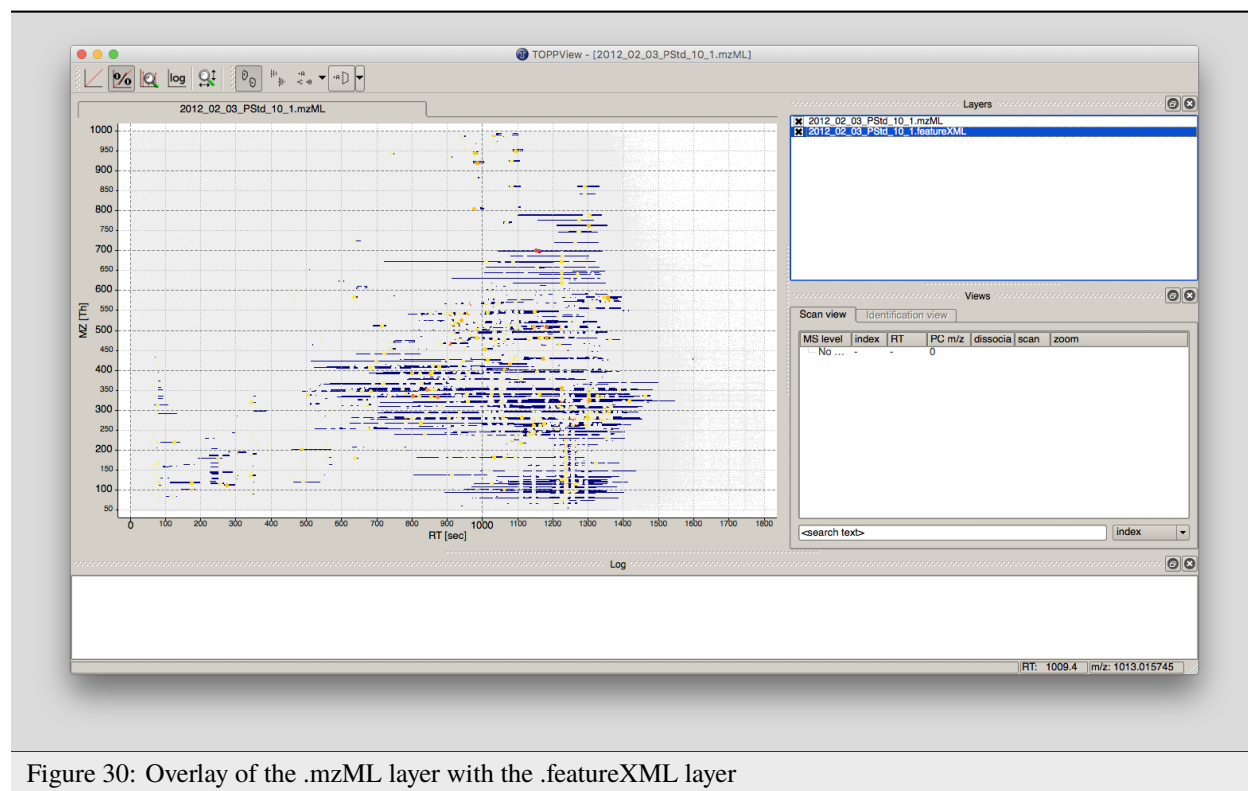


Figure 30: Overlay of the .mzML layer with the .featureXML layer

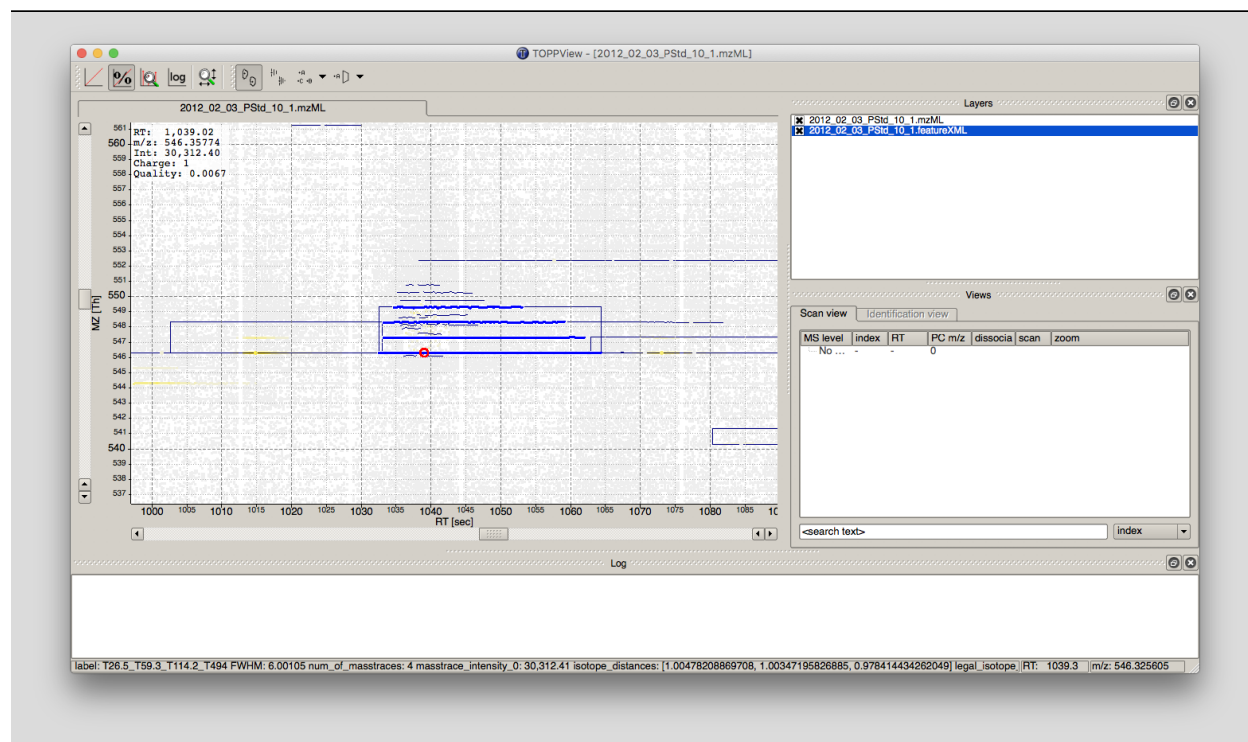


Figure 31: Zoom of the overlay of the .mzML with the .featureXML layer. Here the individual isotope traces (blue lines) are assembled into a feature here shown as convex hull (rectangular box).

The workflow can be extended for multi-file analysis, here an **Input Files** node is to be used instead of the **Input File** node. In front of the **FeatureFinderMetabo**, a **ZipLoopStart** and behind **ZipLoopEnd** has to be used, since **FeatureFinderMetabo** will analysis on file to file bases.

To facilitate the collection of features corresponding to the same compound ion across different samples, an alignment of the samples' feature maps along retention time is often helpful. In addition to local, small-scale elution differences, one can often see constant retention time shifts across large sections between samples. We can use linear transformations to correct for these large scale retention differences. This brings the majority of corresponding compound ions close to each other. Finding the correct corresponding ions is then faster and easier, as we don't have to search as far around individual features.

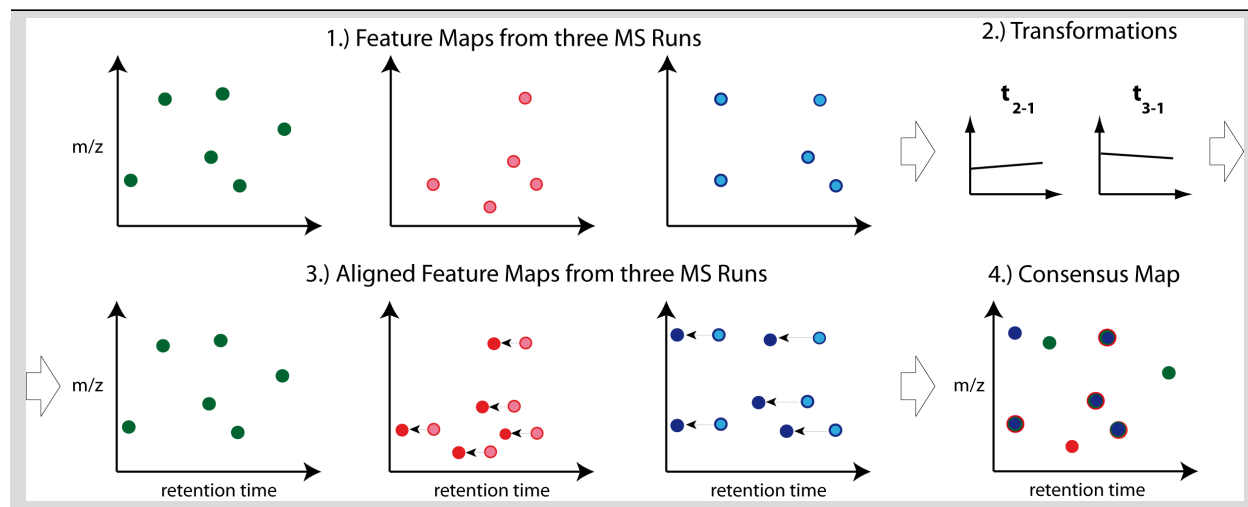


Figure 32: The first feature map is used as a reference to which other maps are aligned. The calculated transformation brings corresponding features into close retention time proximity. Linking of these features form a so-called consensus features of a consensus map.

- 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
<code>algorithm → max_num_peaks_considered</code>	1
<code>algorithm → superimposer → mz_pair_max_distance</code>	0.005
<code>algorithm → superimposer → num_used_points</code>	10000
<code>algorithm → pairfinder → distance_RT → max_difference</code>	20.0
<code>algorithm → pairfinder → distance_MZ → max_difference</code>	20.0
<code>algorithm → pairfinder → distance_MZ → unit</code>	ppm

MapAlignerPoseClustering provides an algorithm to align the retention time scales of multiple input files, correcting shifts and distortions between them. Retention time adjustment may be necessary to correct for chromatography differences e.g. before data from multiple LC-MS runs can be combined (feature linking). The alignment algorithm implemented here is the pose clustering algorithm.

The parameters change the behavior of **MapAlignerPoseClustering** as follows:

- **max_num_peaks_considered**: The maximal number of peaks/features to be considered per map. To use all, set this parameter to -1.
- **mz_pair_max_distance**: Maximum of m/z deviation of corresponding elements in different maps. This condition applies to the pairs considered in hashing.

- **num_used_points**: Maximum number of elements considered in each map (selected by intensity). Use a smaller number to reduce the running time and to disregard weak signals during alignment.
- **distance_RT** → **max_difference**: Features that have a larger RT difference will never be paired.
- **distance_MZ** → **max_difference**: Features that have a larger m/z difference will never be paired.
- **distance_MZ** → **unit**: Unit used for the parameter distance_MZ max_difference, either Da or ppm.

The next step after retention time correction is the grouping of corresponding features in multiple samples. In contrast to the previous alignment, we assume no linear relations of features across samples. The used method is tolerant against local swaps in elution order.

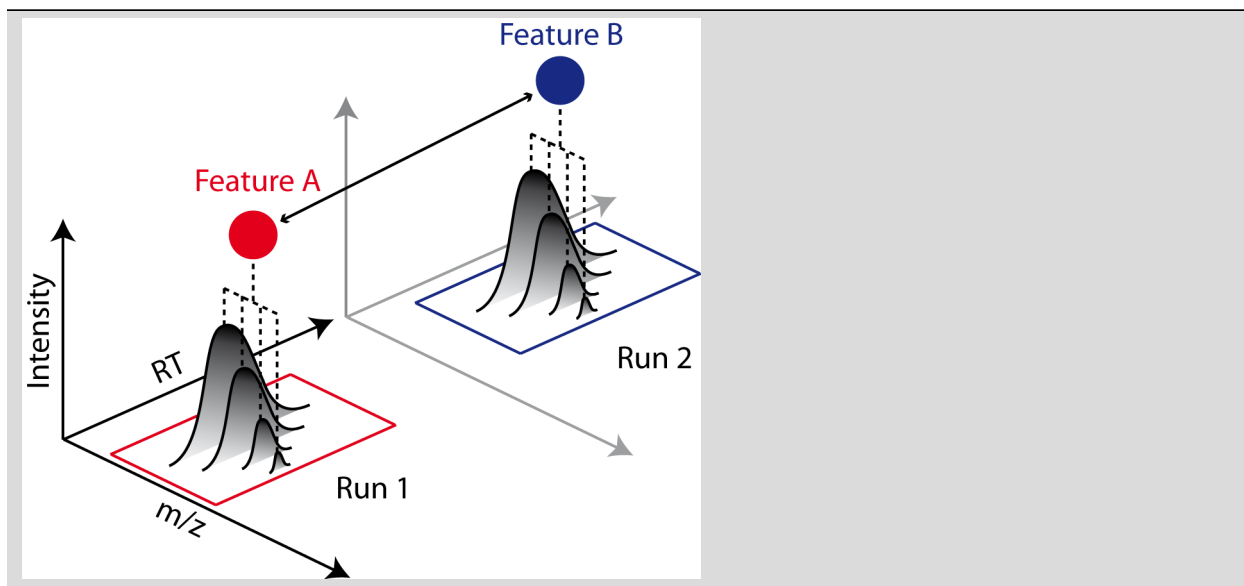


Figure 33: Features A and B correspond to the same analyte. The linking of features between runs (indicated by an arrow) allows comparing feature intensities.

- After the **MapAlignerPoseClustering** node, add a **FeatureLinkerUnlabeledQT** node (**Community Nodes > OpenMS>Map Alignment**) and adjust the following settings:

parameter	value
<i>algorithm</i> → <i>distance_RT</i> → <i>max_difference</i>	40
<i>algorithm</i> → <i>distance_MZ</i> → <i>max_difference</i>	20
<i>algorithm</i> → <i>distance_MZ</i> → <i>unit</i>	ppm

The parameters change the behavior of **FeatureLinkerUnlabeledQT** as follows (similar to the parameters we adjusted for **MapAlignerPoseClustering**):

- **distance_RT** → **max_difference**: Features that have a larger RT difference will never be paired.
- **distance_MZ** → **max_difference**: Features that have a larger m/z difference will never be paired.
- **distance_MZ** → **unit**: Unit used for the parameter distance_MZ max_difference, either Da or ppm.
- After the **FeatureLinkerUnlabeledQT** node, 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.

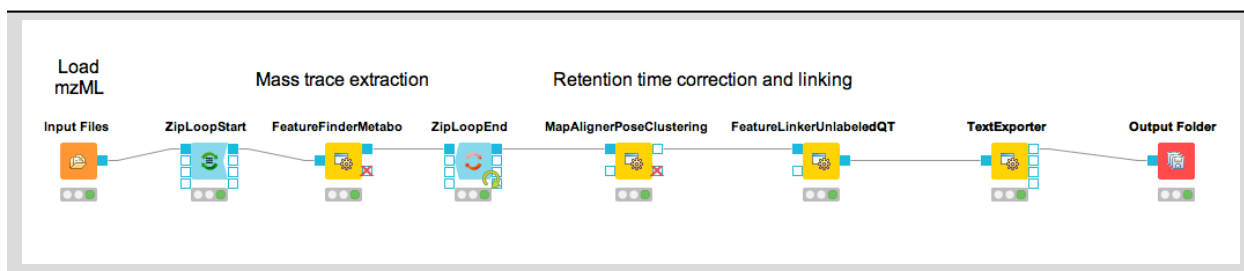


Figure 34: Label-free quantification workflow for metabolites.

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 34. In the following section we will try to identify those metabolites.

The **FeatureLinkerUnlabeledQT** output can be visualized in TOPPView on top of the input and output of the **FeatureFinderMetabo** (see Fig 35).

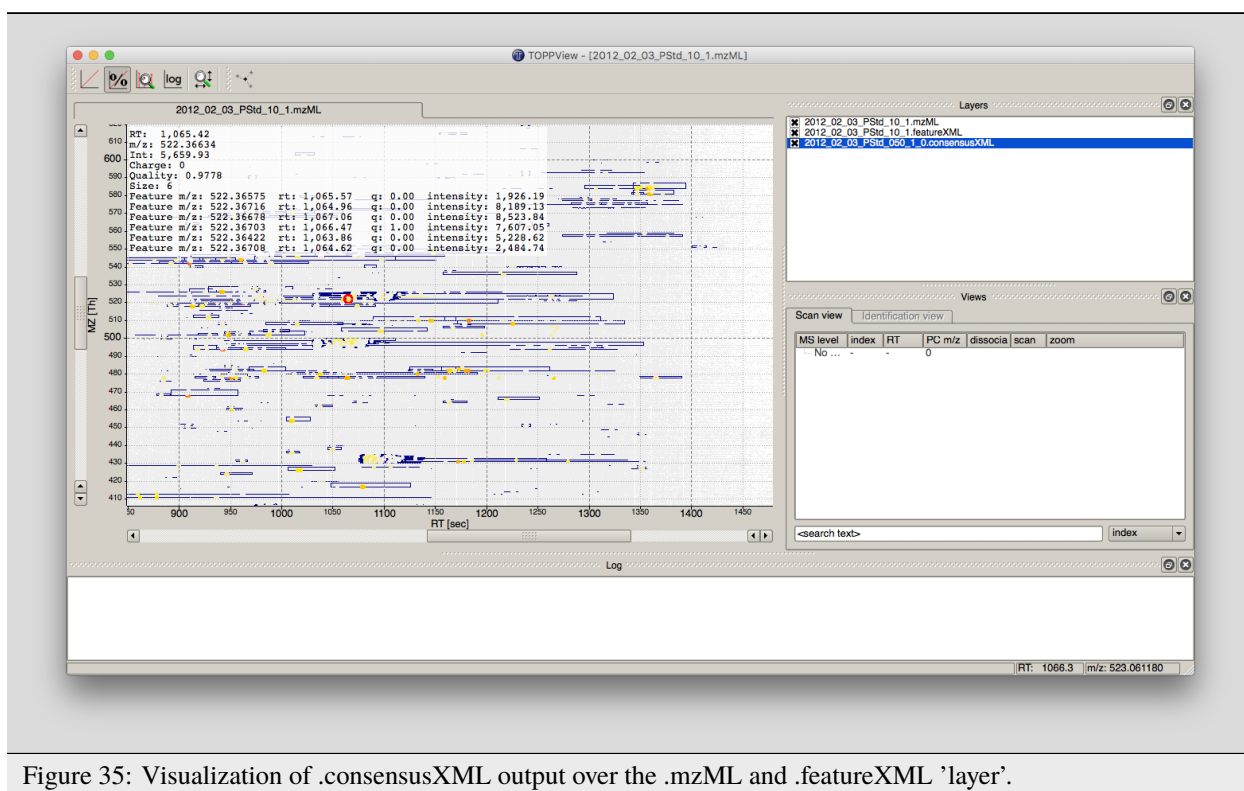


Figure 35: Visualization of .consensusXML output over the .mzML and .featureXML 'layer'.

Basic metabolite identification

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 multiple tools, including search by mass: the **AccurateMassSearch** node searches observed masses against the Human Metabolome Database (HMDB)^{14, 15, 16}. We start with the workflow from the previous section (see Figure 34).

- Add a **FileConverter** node (**Community Nodes > OpenMS > File Handling**) and connect the output of the **FeatureLinkerUnlabeledQT** to the incoming port.
- Open the Configure dialog of the **FileConverter** node and select the tab **OutputTypes**. In the drop down list for **FileConverter.1.out** select **featureXML**.
- Add an **AccurateMassSearch** node (**Community Nodes > OpenMS > Utilities**) and connect the output of the **FileConverter** node to the first port of the **AccurateMassSearch** node.
- Add four **Input File** nodes and configure them with the following files:
 - **Example_DataMetabolomicsdatabasesPositiveAdducts.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.
 - **Example_DataMetabolomicsdatabasesNegativeAdducts.tsv** This file specifies the list of adducts that are considered in the negative mode analogous to the positive mode.
 - **Example_DataMetabolomicsdatabasesHMDBMappingFile.tsv** This file contains information from a metabolite database in this case from HMDB. It has three (or more) tab-separated columns: mass, formula, and identifier(s). This allows for an efficient search by mass.
 - **Example_DataMetabolomicsdatabasesHMDB2StructMapping.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** node.

The result of the **AccurateMassSearch** node is in the mzTab format¹⁷ 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 36.

¹⁴ 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. 69

¹⁵ 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. 69

¹⁶ 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. 69

¹⁷ 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. 69

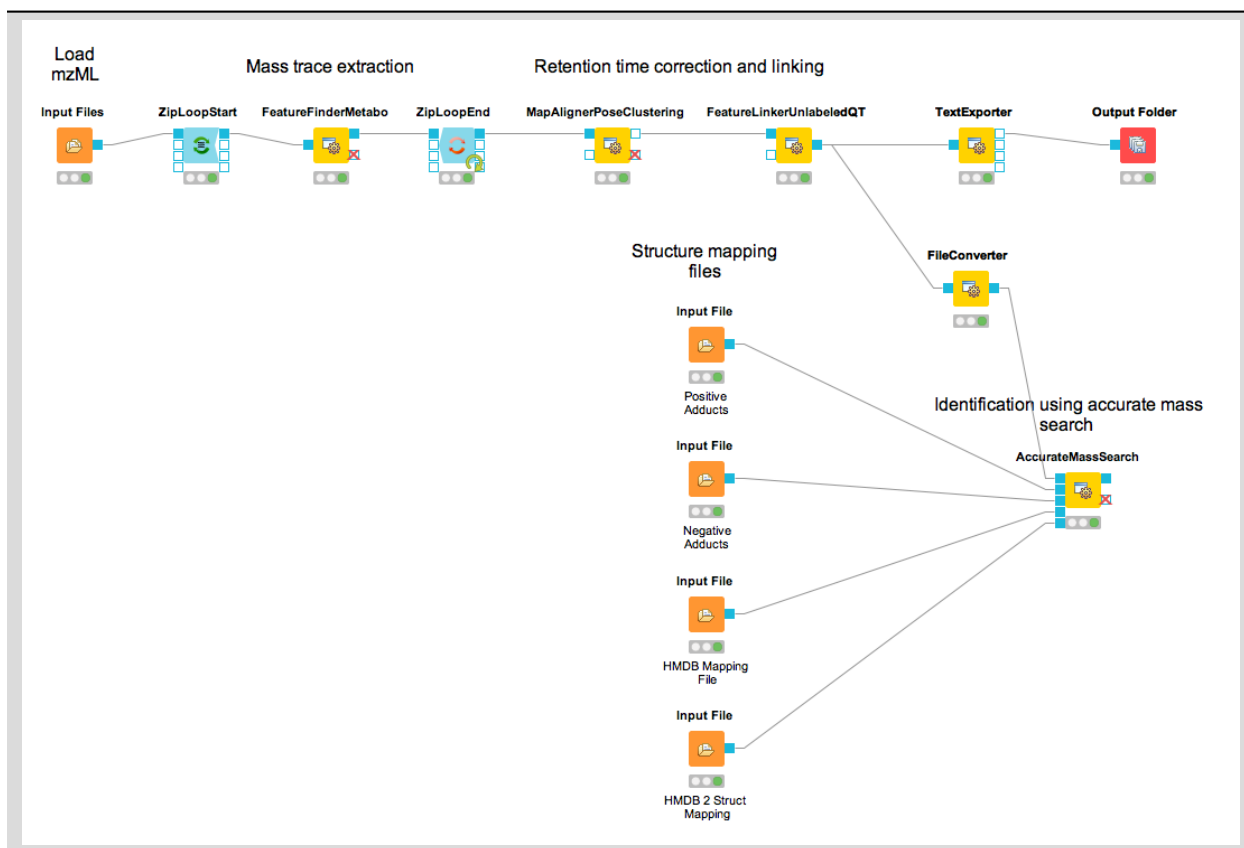


Figure 36: Label-free quantification and identification workflow for metabolites.

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 process only KNIME tables. To convert these files into KNIME tables we need two different nodes. For the **AccurateMassSearch** results, we use the **MzTabReader** node (**Community Nodes > OpenMS > Conversion > mzTab**) and its **Small Molecule Section** port. 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. The retention time values are exported as a list using the **MzTabReader** based on the current PSI-Standard. This has to be parsed using the **SplitCollectionColumn**, which outputs a "Split Value 1" based on the first entry in the retention time list, which has to be renamed to retention time using the **ColumnRename**. You can now combine both tables using the **Joiner** node (**Manipulation > Column > Split & Combine**) and configure it to match the m/z and retention time values of the respective tables. The full workflow is shown in Figure 37.

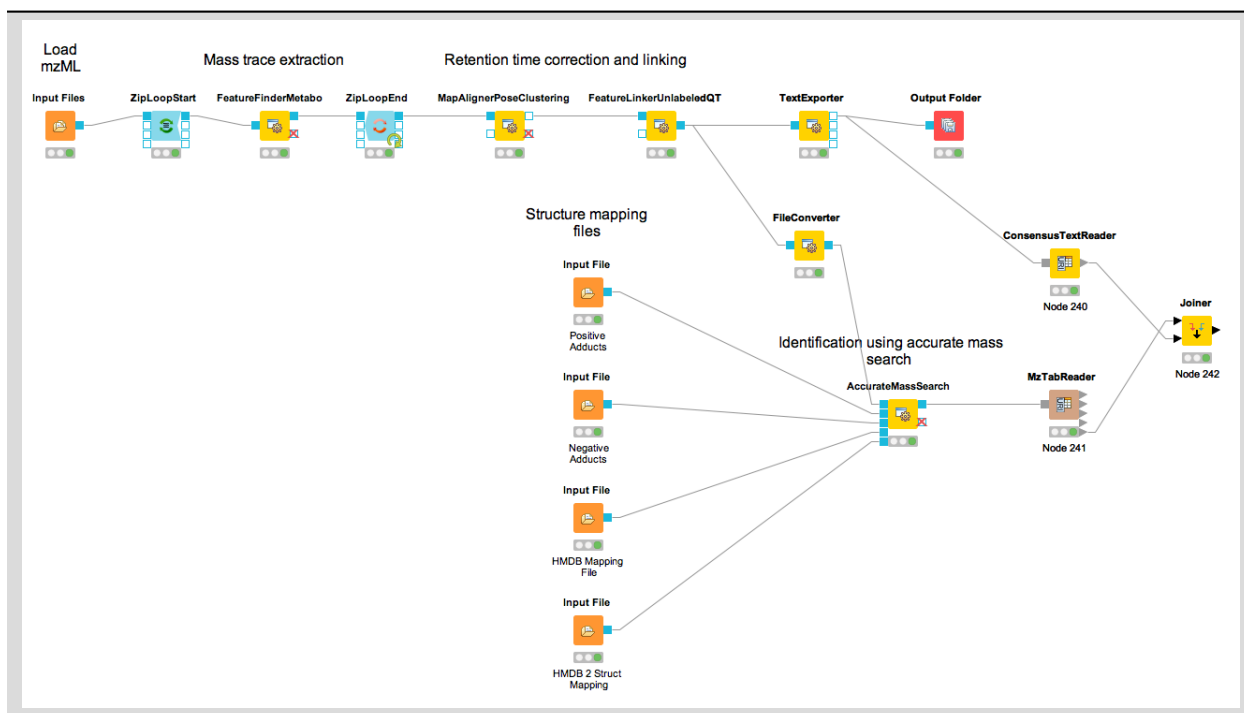


Figure 37: Label-free quantification and identification workflow for metabolites that loads the results into KNIME and joins the tables.

Adduct grouping

Metabolites commonly co-elute as ions with different adducts (e.g., glutathione+H, glutathione+Na) or with charge-neutral modifications (e.g., water loss). Grouping such related ions allows to leverage information across features. For example, a low intensity, single trace feature could still be assigned a charge and adduct due to a matching high-quality feature. Several OpenMS tools, such as **AccurateMassSearch**, can use this information to, for example, narrow down candidates for identification.

For this grouping task, we provide the **MetaboliteAdductDecharger** node. Its method explores the combinatorial space of all adduct combinations in a charge range for optimal explanations. Using defined adduct probabilities, it assigns co-eluting features having suitable mass shifts and charges those adduct combinations which maximize overall ion probabilities.

The tool works natively with featureXML data, allowing the use of reported convex hulls. On such a single-sample level, co-elution settings can be chosen more stringently, as ionization-based adducts should not influence the elution time: Instead, elution differences of related ions should be due to slightly differently estimated times for their feature centroids.

Alternatively, consensusXML data from feature linking can be converted for use, though with less chromatographic information. Here, the elution time averaging for features linked across samples, motivates wider co-elution tolerances.

The two main tool outputs are a consensusXML file with compound groups of related input ions, and a featureXML containing the input file but annotated with inferred adduct information and charges.

Options to respect or replace ion charges or adducts allow for example:

- Heuristic but faster, iterative adduct grouping (**MetaboliteAdductDecharger** → **MetaboliteFeatureDeconvolution** → **q_try** set to “feature”) by chaining multiple **MetaboliteAdductDecharger** nodes with growing adduct sets, charge ranges or otherwise relaxed tolerances.

- More specific feature linking (**FeatureLinkerUnlabeledQT** → **algorithm** → **ignore_adduct** set to “false”)

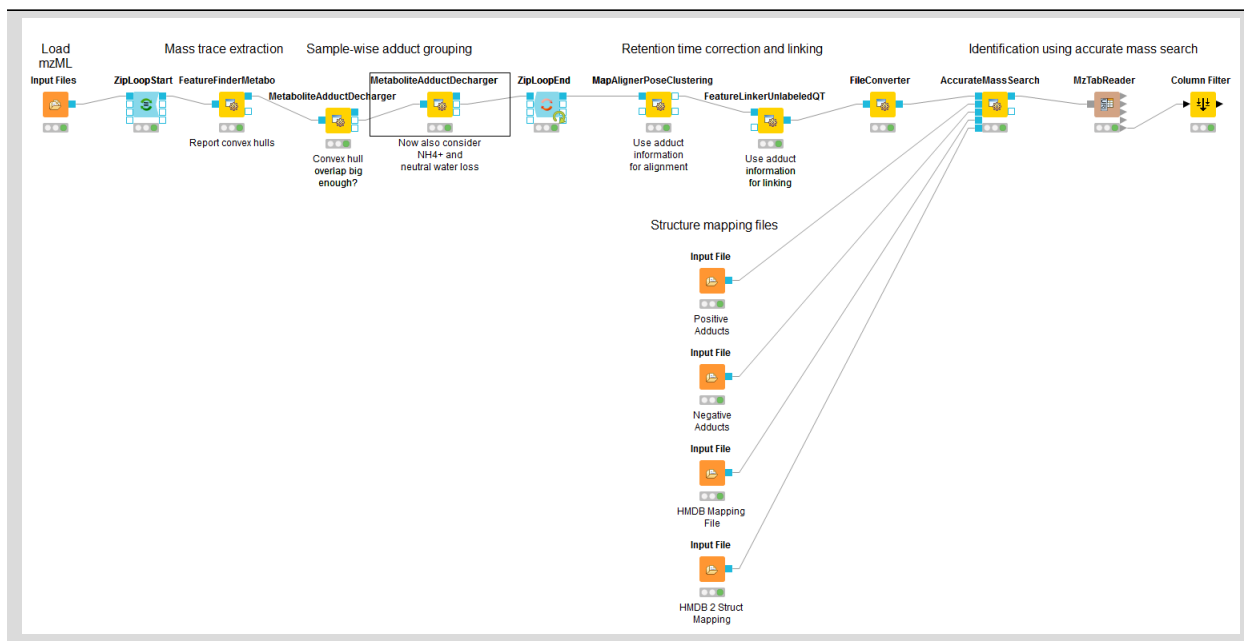


Figure 38: Metabolite Adduct Decharger adduct grouping workflow.

Task

A modified metabolomics workflow with exemplary MetaboliteAdductDecharger use and parameters is provided in WorkflowsMetaboliteAdductGrouping.knwf. Run the workflow, inspect tool outputs and compare **AccurateMassSearch** results with and without adduct grouping.

Visualizing 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 (**Chemistry > Translators**) together with subsequent cheminformatics nodes (e.g. **RDKit From Molecule**(**Community Nodes > RDKit > Converters**)) 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.

Task

Try to compute and visualize the m/z and retention time error of the different feature elements (from the input maps) of each consensus feature. Hint: A nicely configured **Math Formula (Multi Column)** node should suffice.

Spectral library search

Identifying metabolites using only the accurate mass may lead to ambiguous results. In practice, additional information (e.g. the retention time) is used to further narrow down potential candidates. Apart from MS1-based features, tandem mass spectra (MS2) of metabolites provide additional information. In this part of the tutorial, we take a look on how metabolite spectra can be identified using a library of previously identified spectra.

Because these libraries tend to be large we don't distribute them with OpenMS.

Task

Construct the workflow as shown in Fig. 39. Use the file `ExampleData\Metabolomics\datasets\MetaboliteIDSpectraDB-positive.mzML` as input for your workflow. You can use the spectral library from `ExampleData\Metabolomics\database\MetaboliteSpectralDB.mzML` as second input. The first input file contains tandem spectra that are identified by the **MetaboliteSpectralMatcher**. The resulting mzTab file is read back into a KNIME table. The retention time values are exported as a list based on the current PSI-Standard. This has to be parsed using the **SplitCollectionColumn**, which outputs a "Split Value 1" based on the first entry in the retention time list, which has to be renamed to retention time using the **ColumnRename** before it is stored in an Excel table. Make sure that you connect the **MzTabReader** port corresponding to the Small Molecule Section to the **Excel writer (XLS)**. Please select the "add column headers" option in the **Excel writer (XLS)**.

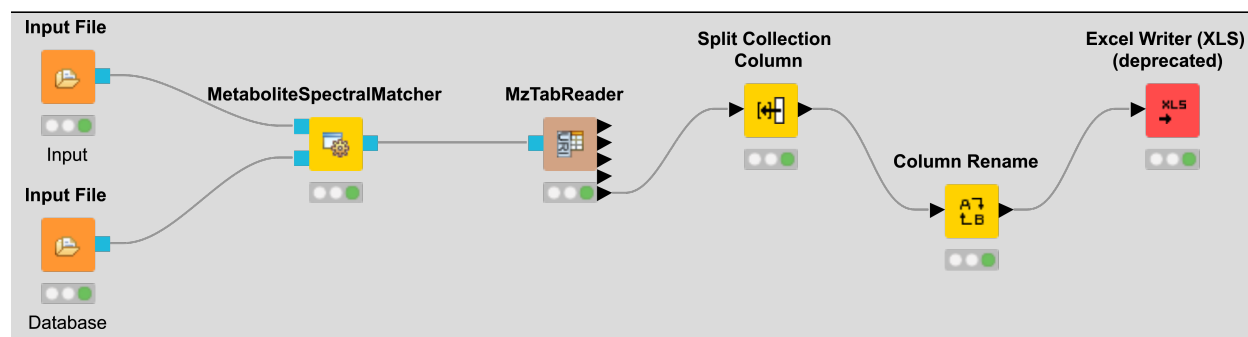


Figure 39: Spectral library identification workflow.

Run the workflow and inspect the output.

Manual validation

In metabolomics, matches between tandem spectra and spectral libraries are manually validated. Several commercial and free online resources exist which help in that task. Some examples are:

- mzCloud contains only spectra from Thermo Orbitrap instruments. The webpage requires Microsoft Silverlight which currently does not work in modern browsers (see the following [link](#)).
- MassBank North America (MoNA) has spectra from different instruments but falls short in number of spectra (compared to Metlin and mzCloud). See the following [link](#).
- METLIN includes 961,829 molecules ranging from lipids, steroids, metabolites, small peptides, carbohydrates, exogenous drugs and toxicants. In total over 14,000 metabolites.

Here, we will use METLIN to manually validate metabolites.

Task

Check in the .xlsx output from the Excel writer (XLS) if you can find glutathione. Use the retention time column to find the spectrum in the mzML file. Here open the file in the Example_DataMetabolomicsdatasetsMetaboliteIDSpectraDBpositive.mzML in TOPPView. The MSMS spectrum with the retention time of 67.6 s is used as example. The spectrum can be selected based on the retention time in the scan view window. Therefore the MS1 spectrum with the retention time of 66.9 s has to be double clicked and the MSMS spectra recorded in this time frame will show up. Select the tandem spectrum of Glutathione, but do not close TOPPView, yet.

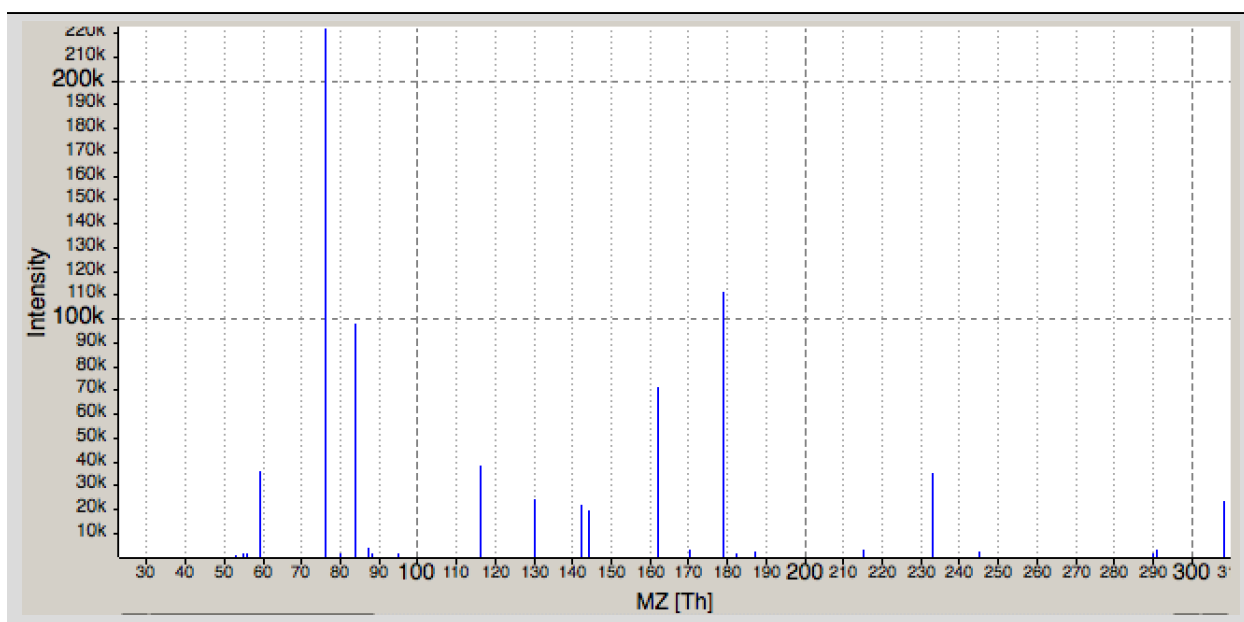


Figure 40: Tandem spectrum of glutathione. Visualized in TOPPView.

Task

On the METLIN homepage search for **Name** Glutathione using the **Advanced Search**. See the [link](#). Note that free registration is required. Which collision energy (and polarity) gives the best (visual) match to your experimental spectrum in TOPPView? Here you can compare the fragmentation patterns in both spectra shown by the Intensity or relative Intensity, the m/z of a peak and the distance between peaks. Each distance between two peaks corresponds to a fragment of elemental composition (e.g., NH₂ with the charge of one would have mass of two peaks of 16.023 Th).

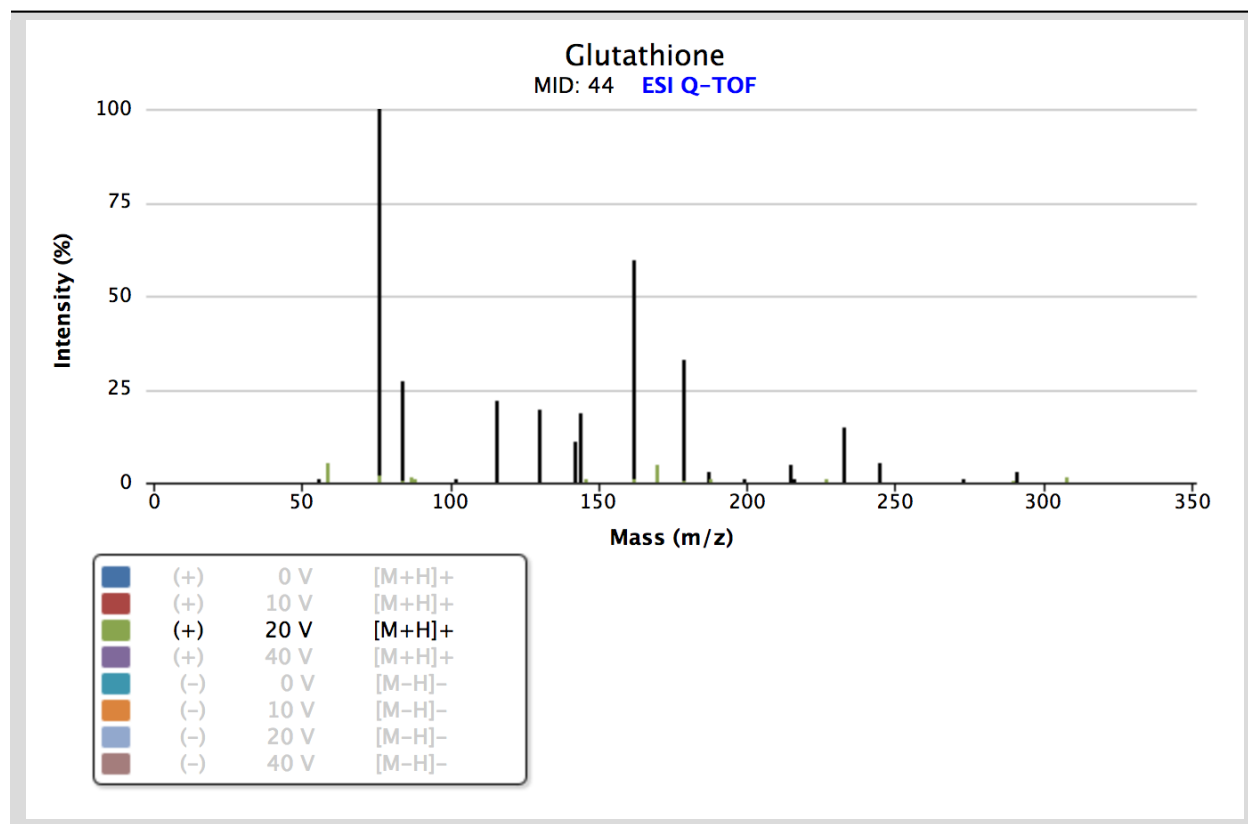


Figure 41: Tandem spectrum of glutathione. Visualized in Metlin. Note that several fragment spectra from varying collision energies are available.

De novo identification

Another method for MS2 spectra-based metabolite identification is de novo identification. This approach can be used in addition to the other methods (accurate mass search, spectral library search) or individually if no spectral library is available. In this part of the tutorial, we discuss how metabolite spectra can be identified using de novo tools. To this end, the tools SIRIUS and CSI:FingerID^(18,19,20) were integrated in the OpenMS Framework as **SiriusAdapter**. SIRIUS uses isotope pattern analysis to detect the molecular formula and further analyses the fragmentation pattern of a compound using fragmentation trees. CSI:FingerID is a method for searching a fingerprint of a small molecule (metabolite) in a molecular structure database. The node **SiriusAdapter** is able to work in different modes depending on the provided input.

- Input: mzML - **SiriusAdapter** will search all MS2 spectra in a map.
- Input: mzML, featureXML (FeatureFinderMetabo) - **SiriusAdapter** can use the provided feature information to reduce the search space to valid features with MS2 spectra. Additionally it can use the isotopic trace information.
- Input: mzML, featureXML (FeatureFinderMetabo / MetaboliteAdductDecharger / AccurateMassSearch) - **SiriusAdapter** can use the feature information as mentioned above together with feature adduct information from adduct grouping or previous identification.

¹⁸ S. Böcker, M. C. Letzel, Z. Lipták, and A. Pervukhin, SIRIUS: Decomposing isotope patterns for metabolite identification, *Bioinformatics* 25(2), 218–224 (2009), doi:10.1093/bioinformatics/btn603. 75

¹⁹ S. Böcker and K. Dührkop, Fragmentation trees reloaded, *J. Cheminform.* 8(1), 1–26 (2016), doi:10.1186/s13321-016-0116-8. 75

²⁰ K. Dührkop, H. Shen, M. Meusel, J. Rousu, and S. Böcker, Searching molecular structure databases with tandem mass spectra using CSI:FingerID, *Proc. Natl. Acad. Sci.* 112(41), 12580–12585 (oct 2015), doi:10.1073/pnas.1509788112. 75

By using a mzML and featureXML, SIRIUS gains a lot of additional information by using the OpenMS tools for preprocessing.

Task

Construct the workflow as shown in Fig. 42. Example_DataMetabolomicsdatasets Use the file `MetaboliteDeNovoID.mzML` as input for your workflow.

Below we show an example workflow for de novo identification (Fig. 42). Here, the node **FeatureFinderMetabo** is used for feature detection to annotate analytes in mz, rt, intensity and charge. This is followed by adduct grouping, trying to assess possible adducts based on the feature space using the **MetaboliteAdductDecharger**. In addition, the **HighResPrecursorMassCorrector** can use the newly generated feature information to map MS2 spectra, which were measured on one of the isotope traces to the monoisotopic precursor. This helps with feature mapping and analyte identification in the **SiriusAdapter** due to the usage of additional MS2 spectra that belong to a specific feature.

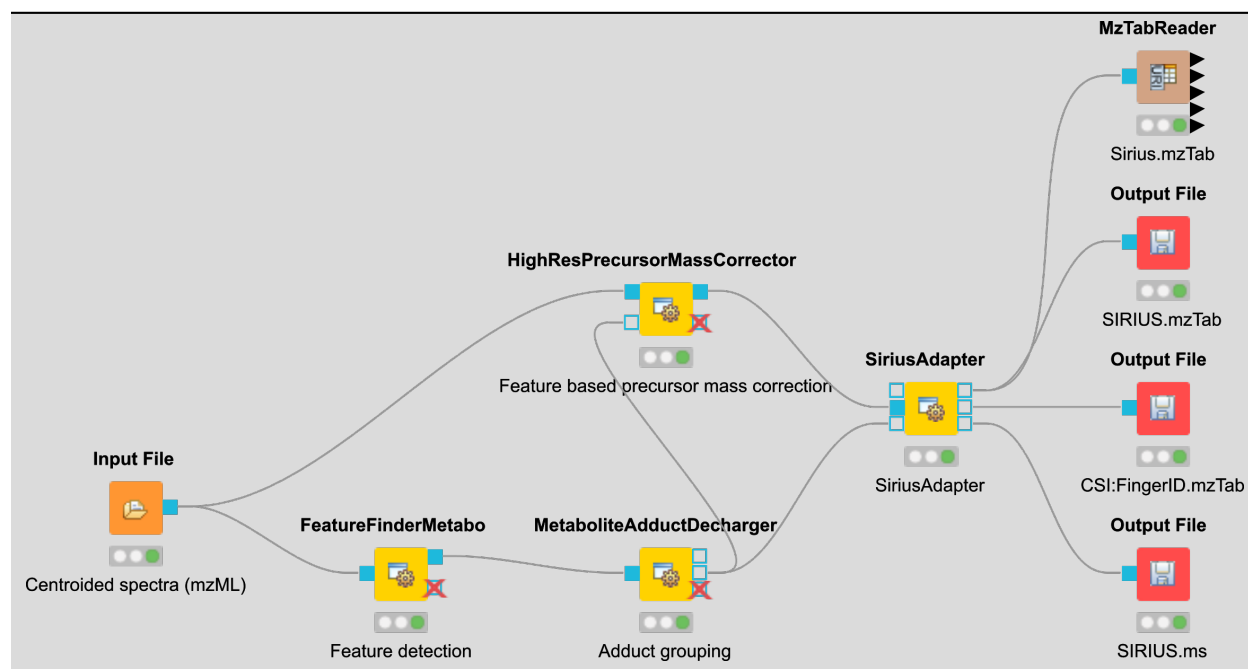


Figure 42: *De novo* identification workflow

Run the workflow and inspect the output.

The output consists of two mzTab files and an internal .ms file. One mzTab for SIRIUS and the other for the CSI:FingerID. These provide information about the chemical formula, adduct and the possible compound structure. The information is referenced to the spectrum used in the analysis. Additional information can be extracted from the **SiriusAdapter** by setting an "out_workspace_directory". Here the SIRIUS workspace will be provided after the calculation has finished. This workspace contains information about annotated fragments for each successfully explained compound.

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 metaboliteID.knwf 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? Note that some basic R knowledge is required to fully understand the processing in **R Snippet** nodes.

Signal processing and data preparation for identification

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

Data preparation for quantification

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(knime.in)` or `cat(knime.in)` to the Console pane.

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 log2 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. 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%).

Interactive visualization

KNIME supports multiple nodes for interactive visualization with interrelated output. The nodes used in this part of the workflow exemplify this concept. They further demonstrate how figures with data dependent customization can be easily realized using basic KNIME nodes. Several simple operations are concatenated in order to enable an interactive volcano plot.

- We first log-transform fold changes and p-values in the **R Snippet** node. We then append columns noting interesting features (concerning fold change and p-value).
- With this information, we can use various Manager nodes (**Views > Property**) to emphasize interesting data points. The configuration dialogs allow us to select columns to change color, shape or size of data points dependent on the column values.
- The **Scatter Plot** node (from the **Views** repository) enables interactive visualization of the logarithmized values as a volcano plot: the log-transformed values can be chosen in the 'Column Selection' tab of the plot view. Data points can be selected in the plot and highlighted via the menu option. The highlighting transfers to all other interactive nodes connected to the same data table. In our case, selection and the highlighting will also occur in the **Interactive Table** node (from the **Views** repository).
- Output of the interactive table can then be filtered via the "HiLite" menu tab. For example, we could restrict shown rows to points highlighted in the volcano plot.

Task

Inspect the nodes of this section. Customize your visualization and possibly try to visualize other aspects of your data.

Advanced visualization

R Dependencies: This section requires that the R packages `ggplot2` and `ggfortify` are both installed. `ggplot2` is part of the KNIME R Statistics Integration (Windows Binaries) which should already be installed via the full KNIME installer, `ggfortify` however is not. In case that you use an R installation where one or both of them are not yet installed, add an **R Snippet** node and double-click to configure. In the R Script text editor, enter the following code:

```
#Include the next line if you also have to install ggplot2:
install.packages("ggplot2")

#Include the following lines to install ggfortify:
install.packages("ggfortify")

library(ggplot2)
library(ggfortify)
```

You can remove the:

```
install.packages
```

commands once it was successfully installed.

Even though the basic capabilities for (interactive) plots in KNIME are valuable for initial data exploration, professional looking depiction of analysis results often relies on dedicated plotting libraries. The statistics language R supports the addition of a large variety of packages, including packages providing extensive plotting capabilities. This part of the workflow shows how to use R nodes in KNIME to visualize more advanced figures. Specifically, we make use of different plotting packages to realize heatmaps.

- The used **RView (Table)** nodes combine the possibility to write R snippet code with visualization capabilities inside KNIME. Resulting images can be looked at in the output RView, or saved via the **Image Writer (Port)** node.
- The heatmap nodes make use of the **gplots** library, which is by default part of the R Windows binaries (for full KNIME version 3.1.1 or higher). We again use regular expressions to extract all measured intensity columns for plotting. For clarity, feature names are only shown in the heatmap after filtering by fold changes.

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**). For that, we use an Inner Join 2 with the **Joiner** node. In the dialog of the node, we add two entries for the Joining Columns and for the first column we pick `retention_time` from the top input (i.e. the **AccurateMassSearch** output) and `rt_cf` (the retention time of the consensus features) for the bottom input (the result from the quantification). For the second column you should choose `exp_mass_to_charge` and `mz_cf` respectively to make the joining unique. Note that the workflow needs to be executed up to the previous nodes for the possible selections of columns to appear.

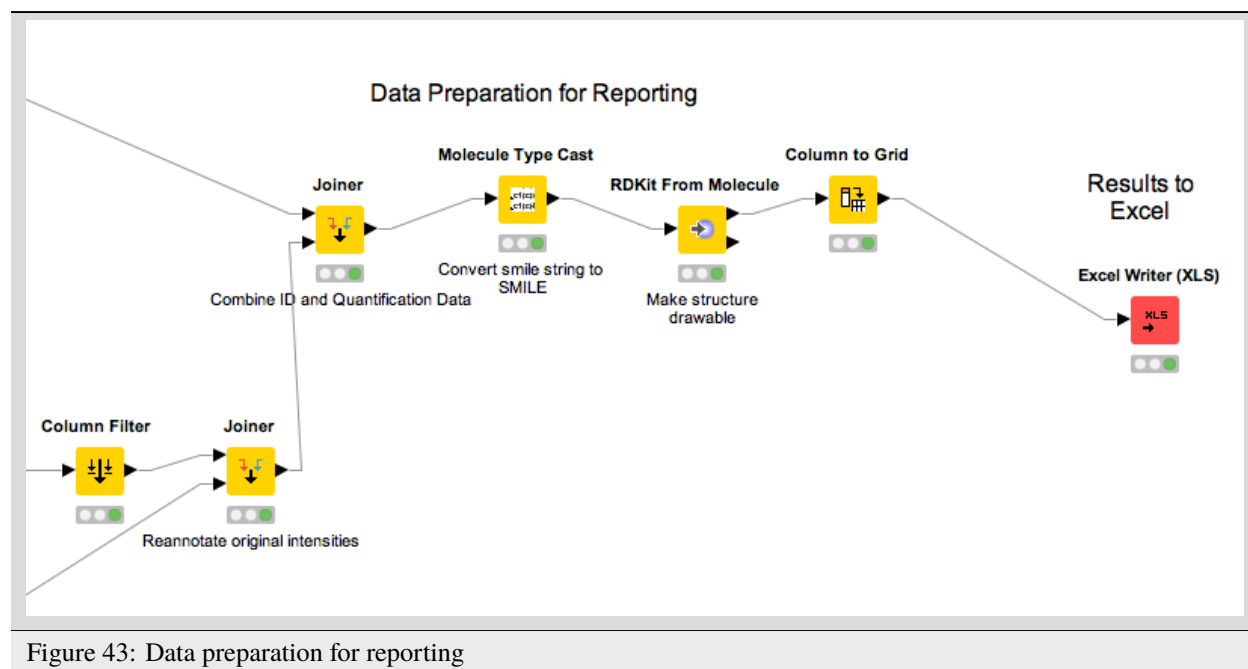


Figure 43: Data preparation for reporting

Question

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

Task

Inspect the output of the join operation after the Molecule Type Cast and RDKit molecular structure generation.

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 this end, 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 export the data to an Excel file (**XLS Writer**).

1.15.7 OpenSWATH

Introduction

OpenSWATH²¹ allows the analysis of LC-MS/MS DIA (data independent acquisition) data using the approach described by Gillet *et al.*²². 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.

Installation of OpenSWATH

OpenSWATH has been fully integrated since OpenMS 1.10^{Page 152, 4, Page 152, 2, 23, 24, 25}.

²¹ 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). 83, 87

²² 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. 83

²³ 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. 83

²⁴ H. L. Röst, T. Sachsenberg, S. Aiche, C. Bielow, H. Weisser, F. Aicheler, S. Andreotti, H.-c. Ehrlich, P. Gutenbrunner, E. Kenar, X. Liang, S. Nahnsen, L. Nilse, J. Pfeuffer, G. Rosenberger, M. Rurik, U. Schmitt, J. Veit, M. Walzer, D. Wojnar, W. E. Wolski, O. Schilling, J. S. Choudhary, L. Malmström, R. Aebersold, K. Reinert, and O. Kohlbacher, OpenMS: a flexible open-source software platform for mass spectrometry data analysis, *Nat. Methods* 13(9), 741–748 (sep 2016), doi:10.1038/nmeth.3959. 83

²⁵ J. Pfeuffer, T. Sachsenberg, O. Alka, M. Walzer, A. Fillbrunn, L. Nilse, O. Schilling, K. Reinert, and O. Kohlbacher, OpenMS - A platform for reproducible analysis of mass spectrometry data, *J. Biotechnol.* 261(February), 142–148 (2017), doi:10.1016/j.jbiotec.2017.05.016. 83

Installation of mProphet

mProphet²⁶ is available as standalone script in External_Tools/mProphet. R and the package MASS are further required to execute mProphet. Please obtain a version for either Windows, Mac or Linux directly from CRAN. PyProphet, a much faster reimplement of the mProphet algorithm is available from PyPI. The usage of pyprophet instead of mProphet is suggested for large-scale applications.

mProphet will be used in this tutorial.

Generating the Assay Library

Generating TraML from transition lists

OpenSWATH requires an assay library to be supplied in the TraML format²⁷. To enable manual editing of transition lists, the TOPP tool **TargetedFileConverter** is available, which uses tab separated files as input. Example datasets are provided in ExampleData/OpenSWATHAssay. Please note that the transition lists need to be named *.tsv.

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

Required Columns: PrecursorMz

The mass-to-charge (m/z) of the precursor ion. (924.539)

ProductMz

The mass-to-charge (m/z) of the product or fragment ion. (728.99)

LibraryIntensity

The relative intensity of the transition. (0.74)

NormalizedRetentionTime

The normalized retention time (or iRT)²⁸ of the peptide. (26.5)

Targeted Proteomics Columns ProteinId

A unique identifier for the protein. (AQUA4SWATH_HMLangeA)

PeptideSequence

The unmodified peptide sequence. (ADSTGTLVITDPTR)

ModifiedPeptideSequence

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

PrecursorCharge

The precursor ion charge. (2)

ProductCharge

The product ion charge. (2)

Grouping Columns: TransitionGroupId

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

²⁶ 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. 83

²⁷ E. W. Deutsch, M. Chambers, S. Neumann, F. Levander, P.-A. Binz, J. Shofstahl, D. S. Campbell, L. Mendoza, D. Ovelheiro, 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. 84

²⁸ 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. 84

TransitionId

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

Decoy

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

PeptideGroupLabel

Which label group the peptide belongs to.

DetectingTransition

Use transition for peak group detection. (1)

IdentifyingTransition

Use transition for peptidoform inference using IPF. (0)

QuantifyingTransition

Use transition to quantify peak group. (1)

For further instructions about generic transition list and assay library generation please see the following [link](#). To convert transitions lists to TraML, use the TargetedFileConverter: Please use the absolute path to your OpenMS installation.

Linux or Mac

On the Terminal:

```
TargetedFileConverter -in OpenSWATH_SGS_AssayLibrary_woDecoy.tsv -out OpenSWATH_SGS_
↳ AssayLibrary_woDecoy.TraML
```

Windows

On the TOPP command:

```
TargetedFileConverter.exe -in OpenSWATH_SGS_AssayLibrary_woDecoy.tsv -out OpenSWATH_SGS_
↳ AssayLibrary_woDecoy.TraML
```

Appending decoys to a TraML file

In addition to the target assays, OpenSWATH requires decoy assays in the library which are later used for classification and error rate estimation. For the decoy generation it is crucial that the decoys represent the targets in a realistic but unnatural manner without interfering with the targets. The methods for decoy generation implemented in OpenSWATH include 'shuffle', 'pseudo-reverse', 'reverse' and 'shift'. To append decoys to a TraML, the TOPP tool **OpenSwathDecoyGenerator** can be used: Please use the absolute path to your OpenMS installation.

Linux or Mac

On the Terminal:

```
OpenSwathDecoyGenerator -in OpenSWATH_SGS_AssayLibrary_woDecoy.TraML -out OpenSWATH_SGS_
↳ AssayLibrary.TraML -method shuffle -switchKR false
```

Windows

On the TOPP command:

```
OpenSwathDecoyGenerator.exe -in OpenSWATH_SGS_AssayLibrary_woDecoy.TraML -out OpenSWATH_
↳ SGS_AssayLibrary.TraML -method shuffle -switchKR false
```

OpenSWATH KNIME

An example KNIME workflow for OpenSWATH is supplied in **Workflows** (Fig. 44). The example dataset can be used for this workflow (filenames in brackets):

1. Open **Workflows**OpenSWATH.knwf 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**. (ExampleDataOpenSWATHassayOpenSWATHiRTAssayLibrary.TraML).
3. Select the SWATH MS data in mzML format as input by double-clicking on node **Input File > SWATH-MS files**. (ExampleDataOpenSWATHdatasplitnapedroL120420x010SW-*.nf.pp.mzML).
4. Select the target peptide assay library in TraML format as input by double-clicking on node **Input Files > Assay Library**. (ExampleDataOpenSWATHassayOpenSWATHSGSAssayLibrary.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 ExampleDataOpenSWATHresult. Please use the absolute path to your R installation and the result file:

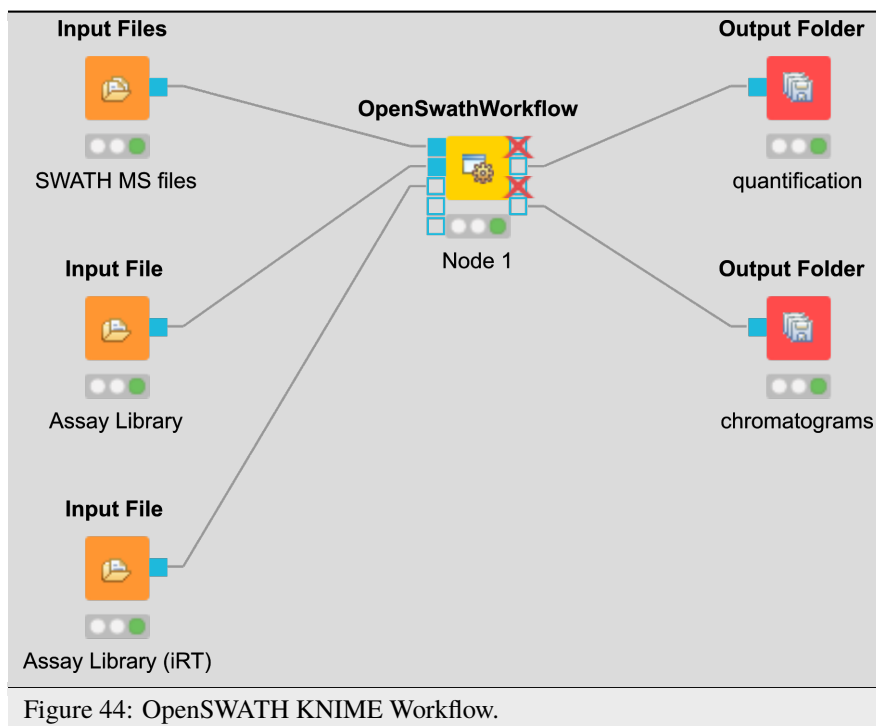
```
R --slave --args bin_dir=../../External_Tools/mProphet/ mquest=OpenSWATH_quant.tsv_
↪ workflow=LABEL_FREE num_xval=5 run_log=FALSE write_classifier=1 write_all_pg=1 < ../../
↪ ../../External_Tools/mProphet/mProphet.R
```

or for Windows:

```
"C:\Program Files\R\R-3.5.1\bin\x86\R.exe" --slave --args bin_dir=../../External_
↪ Tools/mProphet/ mquest=OpenSWATH_quant.tsv 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: OpenSWATHresultmProphetxallxpeakgroups.xls with statistical information available in OpenSWATHresultmProphet.pdf.

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



Additionally, the chromatogram output (.mzML) can be visualized for inspection with TOPPView. For additional instructions on how to use pyProphet instead of mProphet please have a look at the [PyProphet Legacy Workflow](#). If you want to use the SQLite-based workflow in your lab in the future, please have a look [here](#). The SQLite-based workflow will not be part of the tutorial.

From the example dataset to real-life applications

The sample dataset used in this tutorial is part of the larger SWATH MS Gold Standard (SGS) dataset which is described in the publication of Roest *et al.*²¹. 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. Usually SWATH-MS datasets are huge with several gigabyte per run. Especially when complex samples in combination with large assay libraries are analyzed, the TOPP tool based workflow requires a lot of computational resources. Additional information and instruction can be found at the following [link](#).

1.15.8 OpenSWATH for Metabolomics

Introduction

We would like to present an automated DIA/SWATH analysis workflow for metabolomics, which takes advantage of experiment specific target-decoy assay library generation. This allows for targeted extraction, scoring and statistical validation of metabolomics DIA data^{29, 30}.

²⁹ H. L. Röst, G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinović, O. T. Schubert, W. Wolski, B. C. Collins, J. Malmström, L. Malmström, and R. Aebersold, OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data., Nat. Biotechnol. 32(3), 219–23 (2014), doi:10.1038/nbt.2841. 89, 90

³⁰ J. Teleman, H. L. Röst, G. Rosenberger, U. Schmitt, L. Malmström, J. Malmström, and F. Levander, DIANA-algorithmic improvements for analysis of dataindependent acquisition MS data, Bioinformatics 31(4), 555–562 (2015), arXiv: 9808008, doi:10.1093/bioinformatics/btu686. 89, 90

Workflow

The workflow follows multiple steps (see Fig. 45).

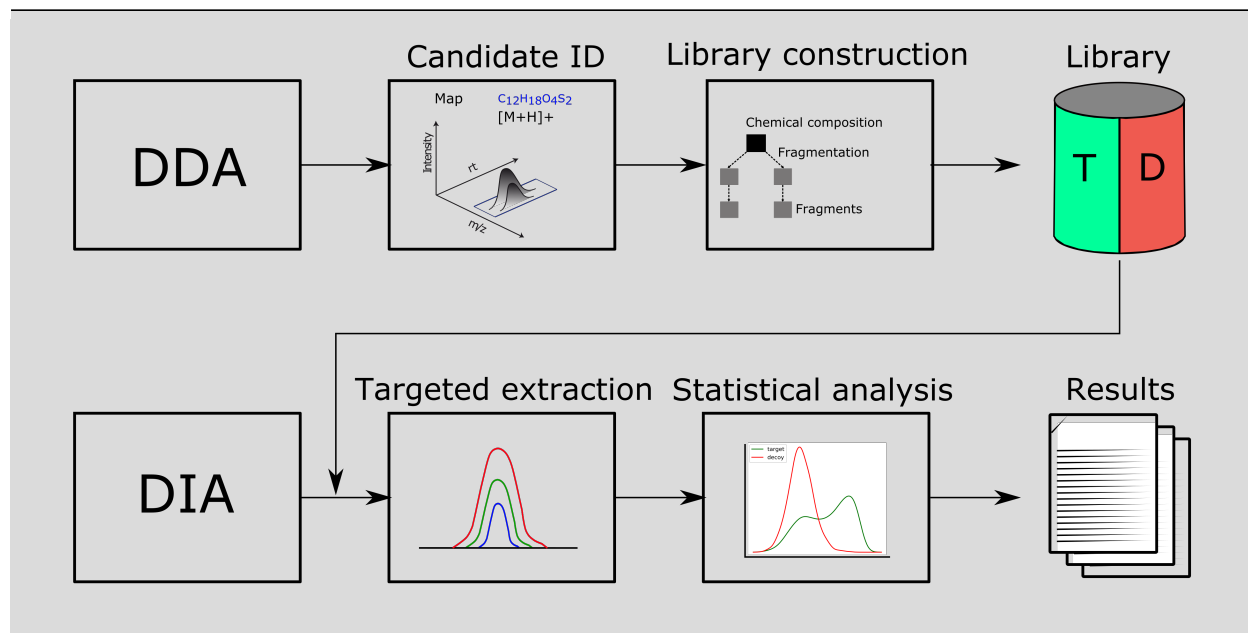


Figure 45: DIAMetAlyzer - pipeline for assay library generation and targeted analysis with statistical validation. DDA data is used for candidate identification containing feature detection, adduct grouping and accurate mass search. Library construction uses fragment annotation via compositional fragmentation trees and decoy generation using a fragmentation tree re-rooting method to create a target-decoy assay library. This library is used in a second step to analyse metabolomics DIA data performing targeted extraction, scoring and statistical validation (FDR estimation).

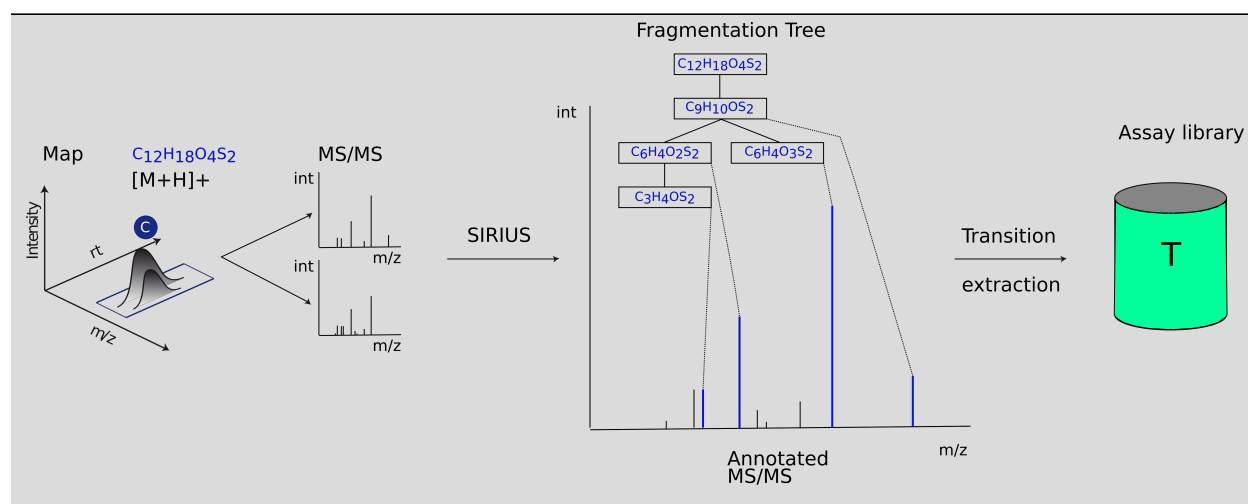


Figure 46: Assay library generation. The results of the compound identification (feature, molecular formula, adduct), with the corresponding fragment spectra for the feature, are used to perform fragment annotation via SIRIUS, using the compositional fragmentation trees. Then, the n highest intensity transitions are extracted and stored in the assay library.

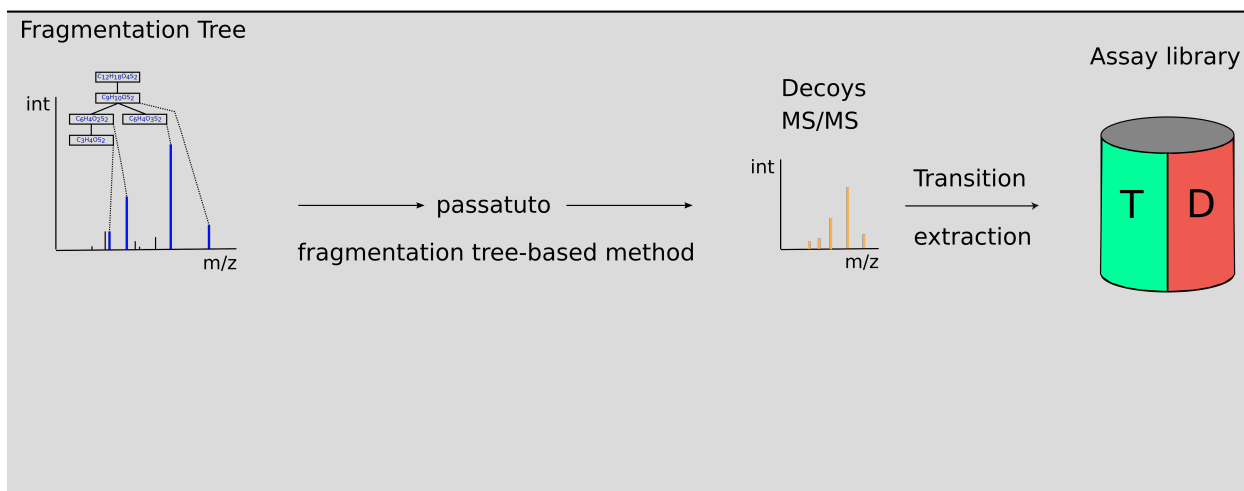


Figure 47: Decoy generation. The compositional fragmentations trees from the step above are used to run the fragmentation tree re-rooting method from Passatutto, generating a compound specific decoy MS2 spectrum. Here, the n highest intensity decoy transitions are extracted and stored in the target-decoy assay library.

- **Candidate identification** Feature detection, adduct grouping and accurate mass search are applied on DDA data.
- **Library construction** The knowledge determined from the DDA data, about compound identification, its potential adduct and the corresponding fragment spectra are used to perform fragment annotation via compositional fragmentation trees using SIRIUS 4³¹. Afterwards transitions, which are the reference of a precursor to its fragment ions are stored in a so-called assay library (Fig. 46). Assay libraries usually contain additional metadata (i.e. retention time, peak intensities). FDR estimation is based on the target-decoy approach³². For the generation of the MS2 decoys, the fragmentation tree-based rerooting method by Passatutto ensures the consistency of decoy spectra (Fig.47)³³. The target-decoy assay library is then used to analyse the SWATH data.
- **Targeted extraction** Chromatogram extraction and peak-group scoring. This step is performed using an algorithm based on OpenSWATH²⁹ for metabolomics data.
- **Statistical validation** FDR estimation uses the PyProphet algorithm^{Page 219, 30}. To prevent overfitting we chose the simpler linear model (LDA) for target-decoy discrimination in PyProphet, using MS1 and MS2 scoring with low correlated scores.

Prerequisites

Apart from the usual KNIME nodes, the workflow uses python scripting nodes. One basic requirement for the installation of python packages, in particular pyOpenMS, is a package manager for python. Using conda as an environment manager allows to specify a specific environment in the KNIME settings (**File>Preferences>KNIME>Python**).

³¹ K. Dührkop, M. Fleischauer, M. Ludwig, A. A. Aksenov, A. V. Melnik, M. Meusel, P. C. Dorrestein, J. Rousu, and S. Böcker, SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information, *Nat. Methods* 16(4), 299–302 (apr 2019), doi:10.1038/s41592-019-0344-8. 89

³² J. E. Elias and S. P. Gygi, Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry, *Nat. Methods* 4(3), 207–214 (Mar. 2007). 89

³³ K. Scheubert, F. Hufsky, D. Petras, M. Wang, L. F. Nothias, K. Dührkop, N. Bandeira, P. C. Dorrestein, and S. Böcker, Significance estimation for large scale metabolomics annotations by spectral matching, *Nat. Commun.* 8(1) (2017), doi:10.1038/s41467-017-01318-5. 89

Windows

We suggest do use a virtual environment for the Python 3 installation on windows. Here you can install miniconda and follow the further instructions.

1. Create new conda python environment.

```
conda create -n py39 python=3.9
```

2. Activate py39 environment.

```
conda activate py39
```

3. Install pip (see above).

4. On the command line:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pandas

python -m pip install -U pyprophet
python -m pip install -U pyopenms
```

macOS

We suggest do use a virtual environment for the Python 3 installation on Mac. Here you can install miniconda and follow the further instructions.

1. Create new conda python environment.

```
conda create -n py39 python=3.9
```

2. Activate py39 environment.

```
conda activate py39
```

3. On the Terminal:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pandas

python -m pip install -U pyprophet
python -m pip install -U pyopenms
```

Linux

Use your package manager apt-get or yum, where possible.

1. Install Python 3.9 (Debian: python-dev, RedHat: python-devel)
2. Install NumPy (Debian/RedHat: python-numpy).
3. Install setuptools (Debian/RedHat: python-setuptools).
4. On the Terminal:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pandas

python -m pip install -U pyprophet
python -m pip install -U pyopenms
```

Benchmark data

For the assay library construction pesticide mixes (Agilent Technologies, Waldbronn, Germany) were measured individually in solvent (DDA). Benchmark DIA samples were prepared by spiking different commercially available pesticide mixes into human plasma metabolite extracts in a 1:4 dilution series, which covers 5 orders of magnitude.

The example data can be found [here](#).

Example workflow

Example workflow for the usage of the DIAMetAlyzer Pipeline in KNIME (see Fig. 48). Inputs are the SWATH-MS data in profile mode (.mzML), a path for saving the new target-decoy assay library, the SIRIUS 4.9.0 executable, the DDA data (.mzML), custom libraries and adducts for **AccurateMassSearch**, the min/max fragment mass-to-charge to be able to restrict the mass of the transitions and the path to the PyProphet executable. The DDA is used for feature detection, adduct grouping, accurate mass search and forwarded to the **AssayGeneratorMetabo**. Here, feature mapping is performed to collect MS2 spectra that belong to a feature. All information collected before (feature, adduct, putative identification, MS2 spectra) are then internally forwarded to SIRIUS. SIRIUS is used for fragment annotation and decoy generation based on the fragmentation tree re-rooting approach. This information is then used to filter spectra/decoys based on their explained intensity (min. 85%). Afterwards internal feature linking is performed which is most important for untargeted experiments using a lot of DDA data to construct the library. The constructed target-decoy assay library is processed with the SWATH-MS data in OpenSWATH. The results are used by PyProphet for scoring and output a list of metabolites with their respective q-value and quantitative information.

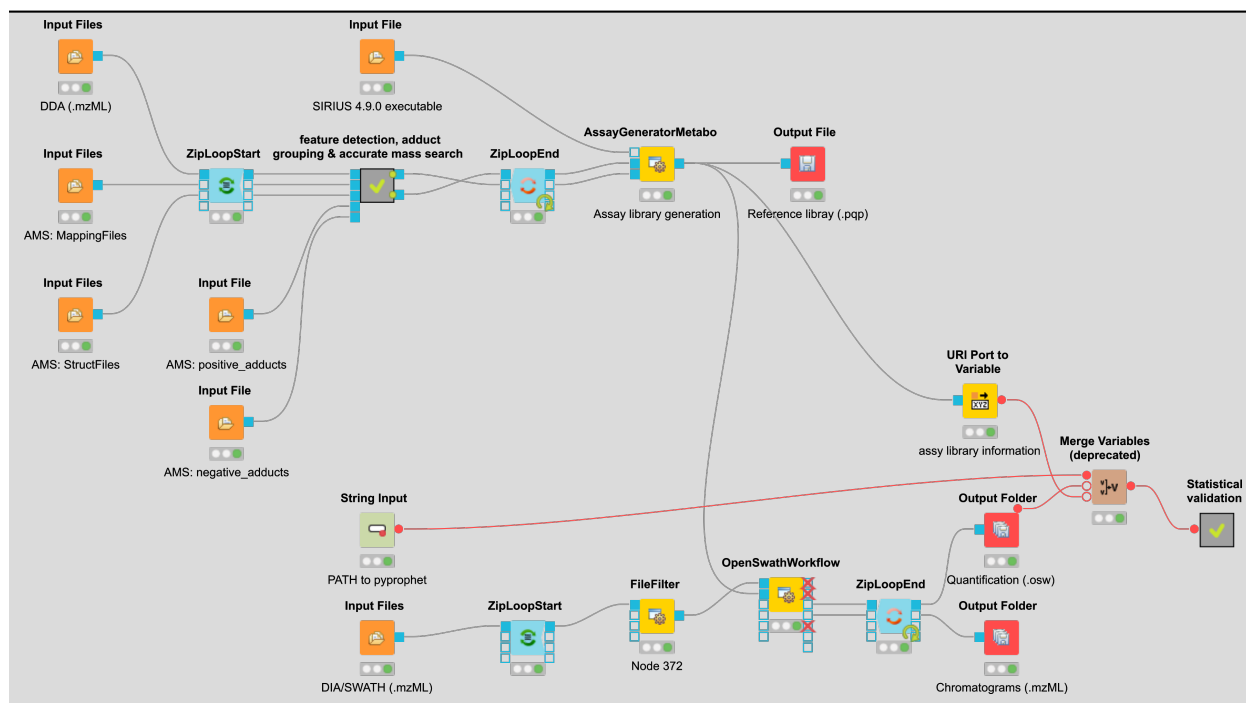


Figure 48: Example workflow for the usage of the DIAMetAlyzer Pipeline in KNIME.

Run the workflow

These steps need to be followed to run the workflow successfully:

- Add DDA Input Files (.mzML).
- Specify SIRIUS 4.9.0 executable.
- Specify library files (mapping, struct) for **AccurateMassSearch**.
- Add positive/negative adducts lists for **AccurateMassSearch**.
- Supply an output path for the SIRIUS workspace in the **AssayGeneratorMetabo**.
- Specify additional paths and variables, such as an output path for the target-decoy assay library and a path to the pyprophet installation as well as decoy fragment mz filter (min/max).
- Input DIA/SWATH files (.mzML).
- Specify output path in the output folders.

You can now run the workflow.

Important parameters

Please have a look at the most important parameters, which should be tweaked to fit your data. In general, OpenMS has a lot of room for parameter optimization to best fit your chromatography and instrumental settings.

FeatureFinderMetabo

parameter	explanation
<i>noise_threshold_int</i>	Intensity threshold below which peaks are regarded as noise.
<i>chrom_fwhm</i>	Expected chromatographic peak width (in seconds).
<i>mass_error_ppm</i>	Allowed mass deviation (in ppm).

MetaboliteAdductDecharger

parameter	explanation
<i>mass_max_diff</i>	Maximum allowed mass tolerance per feature...
<i>potential_adducts</i>	Adducts used to explain mass differences - These should fit to the adduct list specified for AccurateMassSearch.

AccurateMassSearch

parameter	explanation
<i>mass_error_value</i>	Tolerance allowed for accurate mass search.
<i>ionization_mode</i>	Positive or negative ionization mode.

AssayGeneratorMetabo

parameter	explanation
<i>min_transitions</i>	Minimal number of transitions (3).
<i>max_transitions</i>	Maximal number of transitions (3).
min_fragment_mz	Minimal m/z of a fragment ion chosen as a transition
max_fragment_mz	Maximal m/z of a fragment ion chosen as a transition
<i>transitions_threshold</i>	Further transitions need at least x% of the maximum intensity.
fragment_annotation_score	Filters annotations based on the explained intensity of the peaks in a spectrum (0.8).
SIRIUS (internal):	
<i>out_workspace_direct</i>	Output directory for SIRIUS workspace (Fragmentation Trees).
<i>filter_by_num_masstrac</i>	Features have to have at least x MassTraces. To use this parameter <i>feature_only</i> is necessary.
<i>precursor_mass_tolerance</i>	Tolerance window for precursor selection (Feature selection in regard to the precursor).
<i>precursor_rt_tolerance</i>	Tolerance allowed for matching MS2 spectra depending on the feature size (should be around the FWHM of the chromatograms).
<i>profile_elements</i>	Specify the used analysis profile (e.g. qtof). Allowed elements for assessing the putative sumformula (e.g. CHNOP[5]S[8]Cl[1]). Elements found in the isotopic pattern are added automatically, but can be specified nonetheless.
Feature linking (internal):	
ambiguity_resolution_mz_tolerance	M/z tolerance for the resolution of identification ambiguity over multiple files - Feature linking m/z tolerance.
ambiguity_resolution_rt_tolerance	RT tolerance in seconds for the resolution of identification ambiguity over multiple files - Feature linking m/z tolerance.
total_occurrence_filter	Filter compound based on total occurrence in analysed samples.

In case of the **total_occurrence_filter** the value to chose depends on the analysis strategy used. In the instance you are using only identified compounds (**use_known_unknowns**= false) - it will filter based on identified features. This means that even if the feature was detected in e.g. 50% of all samples it might be only identified correctly by accurate mass search in 20% of all samples. Using a **total_occurrence_filter** this specific feature would still be filtered out due to less identifications.

OpenSWATH

parameter	explanation
<i>rt_extraction_window</i>	Extract x seconds around this value.
<i>rt_normalization_factor</i>	Please use the range of your gradient e.g. 950 seconds.

If you are analysing a lot of big DIA mzML files 3-20GB per File, it makes sense to change how OpenSWATH processes the spectra.

parameter	explanation
<i>readOptions</i>	Set cacheWorkingInMemory - will cache the files to disk and read SWATH-by-SWATH into memory
<i>tempDirectory</i>	Set a directory, where cached mzMLs are stored (be aware that this directory can be quite huge depending on the data).

In the workflow pyprophet is called after OpenSWATH, it merges the result files, which allows to get enough data for the model training.

```
pyprophet merge --template path_to_target-decoy_assay_library.pqp --out merged.osw, → ./
↳ *.osw
```

Afterwards, the results are scored using the MS1 and MS2 levels and filter for metabolomics scores, which have a low correlation.

```
pyprophet score --in merged.osw --out scored.osw --level ms1ms2 --ss_main_score, → "var_
↳ isotope_correlation_score" --ss_score_filter metabolomics
```

Export the non filtered results:

```
pyprophet export-compound --in scored.osw --out scored + "_pyprophet_nofilter_ms1ms2.
↳ tsv" --max_rs_peakgroup_qvalue 1000.0
```

Please see the workflow for actual parameter values used for the benchmarking dataset.

The workflow can be used without any identification (remove `AccurateMassSearch`). Here, all features (**known_unknowns**) are processed. The assay library is constructed based on the chemical composition elucidated via the fragment annotation (SIRIUS 4). It is also possible to use identified and in addition unknown (non-identified) features, by using **AccurateMassSearch** in combination with the `use_known_unknowns` in the **AssayGeneratorMetabo**.

1.15.9 Untargeted metabolomics preprocessing

The universal workflow for untargeted metabolomics always consists of feature detection in the individual MS sample files and their linkage to consensus features with common m/z and retention time values. In addition, there are optional steps such as adduct detection and annotation of features with associated MS2 spectra. This workflow prepares all the file necessary to do formula and structural annotations via **SiriusAdapter**. Furthermore it prepares all required files to run **GNPSExport**, which generates all files necessary to directly run **GNPS** Feature Based Molecular Networking (FBMN) and Ion Identity Molecular Networking (IIMN).

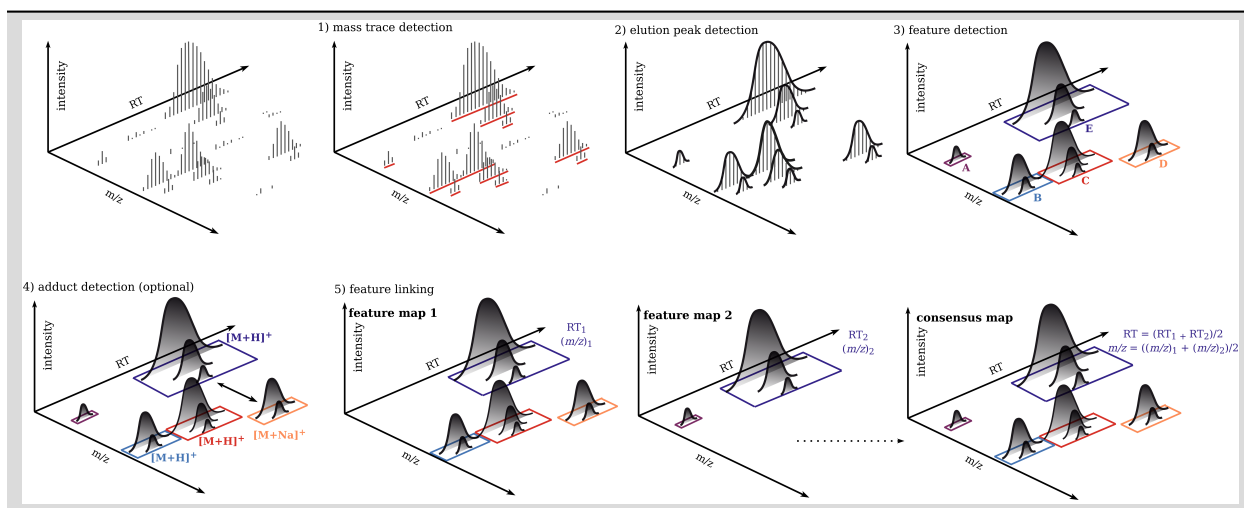


Figure 49: Metabolomics preprocessing steps

If you want to use the example data, download the files `sample1.mzML` and `sample2.mzML`.

For each `mzML` file do mass trace, elution peak and feature detection.

```
FeatureFinderMetabo -in sample1.mzML -out sample1.featureXML -algorithm:common:noise_
↳ threshold_int 10000 -algorithm:mtd:mass_error_ppm 10 -algorithm:ffm:remove_single_
↳ traces true
FeatureFinderMetabo -in sample2.mzML -out sample2.featureXML -algorithm:common:noise_
↳ threshold_int 10000 -algorithm:mtd:mass_error_ppm 10 -algorithm:ffm:remove_single_
↳ traces true
```

Align feature retention times based on the feature map with the highest number of features (reference map).

```
MapAlignerPoseClustering -in sample1.featureXML sample2.featureXML -out aligned_sample1.
↳ featureXML aligned_sample2.featureXML -trafo_out sample1.trafoXML sample2.trafoXML -
↳ algorithm:pairfinder:distance_MZ:max_difference 10.0 -algorithm:pairfinder:distance_
↳ MZ:unit ppm
```

Align `mzML` files alignment based on FeatureMap alignment (optional, only for GNPS).

```
MapRTTransformer -in sample1.mzML -out aligned_sample1.mzML -trafo_in sample1.trafoXML
MapRTTransformer -in sample2.mzML -out aligned_sample2.mzML -trafo_in sample2.trafoXML
```

Map MS2 spectra to features as `PeptideIdentification` objects (optional, only for GNPS). Requires an empty `idXML` file.

```
IDMapper -id empty.idXML -in aligned_sample1.featureXML -spectra:in aligned_sample1.mzML_
↳ -out IDmapped_sample1.featureXML
IDMapper -id empty.idXML -in aligned_sample2.featureXML -spectra:in aligned_sample2.mzML_
↳ -out IDmapped_sample2.featureXML
```

Detect adducts (optional, only for SIRIUS and GNPS Ion Identity Molecular Networking).

```
MetaboliteAdductDecharger -in IDmapped_sample1.featureXML -out_fm adducts_sample1.
↳ featureXML -algorithm:MetaboliteFeatureDeconvolution:potential_adducts "H+:0.6"
```

(continues on next page)

(continued from previous page)

```

↪ "Na:+:0.1" "NH4:+:0.1" "H-10-1:+:0.1" "H-30-2:+:0.1"
MetaboliteAdductDecharger -in IDmapped_sample2.featureXML -out_fm adducts_sample2.
↪ featureXML -algorithm:MetaboliteFeatureDeconvolution:potential_adducts "H:+:0.6"
↪ "Na:+:0.1" "NH4:+:0.1" "H-10-1:+:0.1" "H-30-2:+:0.1"

```

Link features in a ConsensusMap.

```

FeatureLinkerUnlabeledKD -in adducts_sample1.featureXML adducts_sample2.featureXML -out_
↪ Preprocessed.consensusXML -algorithm:link:rt_tol 30.0 -algorithm:link:mz_tol 10.0

```

Export table of metabolic features as tsv file including meta values (e.g. best consensus adduct ion).

```

TextExporter -in Preprocessed.consensusXML -out Features.tsv -consensus:add_metavalues

```

You can recreate this workflow in KNIME. Download the KNIME workflow here. The workflow should look like this:

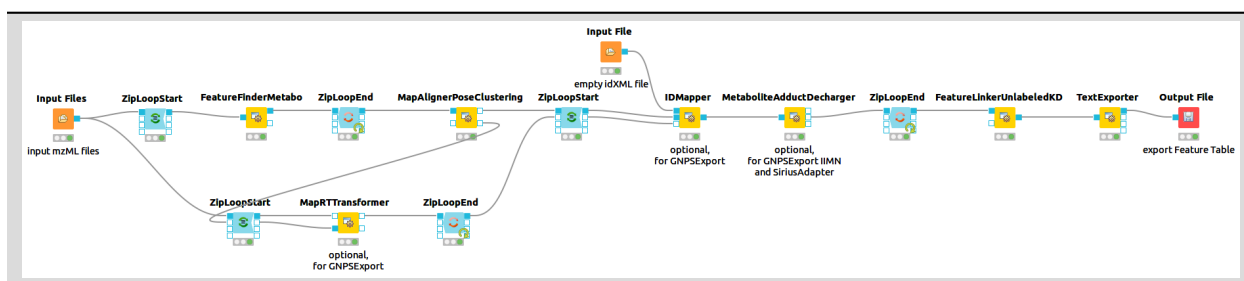


Figure 50: Metabolomics preprocessing workflow in KNIME

1.15.10 An introduction to pyOpenMS

Introduction

pyOpenMS provides Python bindings for a large part of the OpenMS library for mass spectrometry based proteomics and metabolomics. It thus provides access to a featurerich, open-source algorithm library for mass-spectrometry based LC-MS 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](#).

Installation

One basic requirement for the installation of python packages, in particular pyOpenMS, is a package manager for python. We provide a package for [pip](#).

Windows

1. Install [Python 3.9](#).
2. Install [NumPy](#).
3. Install pip (see above).
4. On the command line:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install pyopenms
```

macOS

We suggest do use a virtual environment for the Python 3 installation on Mac. Here you can install miniconda and follow the further instructions.

1. Create new conda python environment.

```
conda create -n py37 python=3.9 anaconda
```

2. Activate py37 environment.

```
source activate py37
```

3. On the Terminal:

```
pip install -U pip
pip install -U numpy
pip install pyopenms
```

Linux

Use your package manager apt-get or yum, where possible.

1. Install Python 3.9 (Debian: python-dev, RedHat: python-devel).
2. Install NumPy (Debian / RedHat: python-numpy).
3. Install setuptools (Debian / RedHat: python-setuptools).
4. On the Terminal:

```
pip install pyopenms
```

IDE with Anaconda integration

If you do not have python installed or do not want to modify your native installation, another possibility is to use an IDE (integrated development environment) with Anaconda integration. Here, we recommend [spyder](#). It comes with Anaconda, which is a package and environment manager. Thus the IDE should be able to run a specific environment independent of your systems python installation. Please execute the installer for your respective platform located in the respective directory for your platform and follow the installation instructions. After installation, the ANACONDA Navigator (Anaconda 3) should be available. Please start the application. To install pyopenms please choose the button "Environments" and click the play symbol of the base environment and "Open Terminal". Update pip and install pyopenms (MacOS, Linux):

```
pip install -U pip
pip install -U numpy
pip install -U pyopenms
```

Update pip and install pyopenms:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U pyopenms
```

Install a local available package:

```
pip install numpy-1.20.0-cp37*.whl
pip install pyopenms-3.0.0-cp37*.whl
or (in case of windows)

python -m pip install -U numpy-1.20.0-cp37*.whl
python -m pip install -U pyopenms-3.0.0-cp37*.whl
```

The local available packages can be found in the directory corresponding to your operating system. Please use the absolute path to the packages for the installation.

Now launch "Spyder" (python IDE) in the home menu.

Build instructions

Instructions on how to build pyOpenMS can be found [online](#).

Scripting with pyOpenMS

A big advantage of pyOpenMS are its scripting capabilities (beyond its application in tool development). Most of the OpenMS datastructure can be accessed using [python](#). Here we would like to give some examples on how pyOpenMS can be used for simple scripting task, such as peptide mass calculation and peptide/protein digestion as well as isotope distribution calculation.

Calculation of the monoisotopic and average mass of a peptide sequence:

```
from pyopenms import *

seq = AASequence.fromString("DFPIANGER")
```

(continues on next page)

(continued from previous page)

```

mono_mass = seq.getMonoWeight(Residue.ResidueType.Full, 0)

average_mass = seq.getAverageWeight(Residue.ResidueType.Full, 0)

print("The masses of the peptide sequence " + seq.toString().decode('utf-8') + " are:")

print("mono: " + str(mono_mass))
print("average: "+ str(average_mass))

```

Enzymatic digest of a peptide/protein sequence:

```

enzyme = "Trypsin"
to_digest = AASequence.fromString("MKWVTFISLLLLFSSAYSRGVFRRDTHKSEIAHRFKDLGE")

after_digest = []

EnzymaticDigest = EnzymaticDigestionLogModel()

EnzymaticDigest.setEnzyme(enzyme)
EnzymaticDigest.digest(to_digest, after_digest)

print("The peptide " + to_digest.toString().decode('utf-8') + " was digested using " +
↳str(EnzymaticDigest.getEnzymeName().decode('utf-8')) + " to:")

for element in after_digest:
    print(element.toString().decode('utf-8'))

```

Use empirical formula to calculate the isotope distribution:

```

from pyopenms import *

methanol = EmpiricalFormula("CH3OH")
water = EmpiricalFormula("H2O")

wm = EmpiricalFormula(water.toString().decode('utf-8') + methanol.toString().decode('utf-
↳8'))

print(wm.toString().decode('utf-8'))
print(wm.getElementalComposition())

isotopes = wm.getIsotopeDistribution( CoarseIsotopePatternGenerator(3) )

for iso in isotopes.getContainer():
    print (iso.getMZ(), ":", iso.getIntensity())

```

For further examples and the pyOpenMS data structures please see the following [link](#).

Tool development with pyOpenMS

Scripting is one side of pyOpenMS, the other is the ability to create Tools using the C++ OpenMS library in the background. In the following section we will create a "ProteinDigester" pyOpenMS Tool. It should be able to read in a fasta file. Digest the proteins with a specific enzyme (e.g. Trypsin) and export an idXML output file. Please see ExampleDatapyopenms for code snippets.

```
usage: ProteinDigester.py [-h] [-in INFILE] [-out OUTFILE] [-enzyme ENZYME]
                        [-min_length MIN_LENGTH] [-max_length MAX_LENGTH]
                        [-missed_cleavages MISSED_CLEAVAGES]

ProteinDigester  In silico digestion of proteins.

optional arguments:
  -h, --help            show this help message and exit
  -in INFILE            An input file containing amino acid sequences [fasta]
  -out OUTFILE          Output digested sequences in idXML format [idXML]
  -enzyme ENZYME        Enzyme used for digestion
  -min_length MIN_LENGTH Minimum length of peptide
  -max_length MAX_LENGTH Maximum length of peptide
  -missed_cleavages MISSED_CLEAVAGES The number of allowed missed_
  ↪cleavages
```

Basics

First, your tool needs to be able to read parameters from the command line and provide a main routine. Here standard Python can be used (no pyOpenMS is required so far).

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

def main(options):

    # test parameter handling

    print(options.infile, options.outfile, options.enzyme, options.min_length, options.
    ↪max_length, options.missed_cleavages)

def handle_args():
    import argparse

    usage = ""

    usage += "\nProteinDigester  In silico digestion of proteins."
```

(continues on next page)

(continued from previous page)

```

parser = argparse.ArgumentParser(description = usage)

parser.add_argument('-in', dest='infile', help='An input file containing amino acid_
↳sequences [fasta]')

parser.add_argument('-out', dest='outfile', help='Output digested sequences in idXML_
↳format [idXML]')

parser.add_argument('-enzyme', dest='enzyme', help='Enzyme used for digestion')

parser.add_argument('-min_length', type=int, dest='min_length', help = 'Minimum_
↳length of peptide')

parser.add_argument('-max_length', type=int, dest='max_length', help='Maximum length_
↳of peptide')

parser.add_argument('-missed_cleavages', type=int, dest='missed_cleavages', help=
↳'The number of allowed missed cleavages')

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

if __name__ == '__main__':

    options = handle_args()
    main(options)

```

Open the Anaconda Terminal and change into the ExampleDatapyopenms directory. Execute the example script.

```

python ProteinDigester_argparse.py -h
python ProteinDigester_argparse.py -in mini_example.fasta -out mini_example_out.idXML -
↳enzyme Trypsin -min_length 6 -max_length 40 -missed_cleavages 1

```

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.

OpenMS has a `ProteaseDB` class containing a list of enzymes which can be used for digestion of proteins. You can add this to the `argparse` code to be able to see the usable enzymes. From this point onward, `pyOpenMS` is required.

```

# from here pyopenms is needed
# get available enzymes from ProteaseDB

all_enzymes = []
p_db=ProteaseDB().getAllNames(all_enzymes)

# concatenate them to the enzyme argument.
parser.add_argument('-enzyme', dest='enzyme', help='Enzymes which can be used for_
↳digestion: ' + ', '.join(map(bytes.decode, all_enzymes)))

```

Loading data structures with pyOpenMS

We already scripted enzymatic digestion with the `AASequence` and `EnzymaticDigest` (see above). To make this even easier, we can use an existing class in OpenMS, called `ProteaseDigestion`.

```
# Use the ProteaseDigestion class
# set the enzyme used for digestion and the number of missed cleavages
digestor = ProteaseDigestion()
digestor.setEnzyme(options.enzyme)
digestor.setMissedCleavages(options.missed_cleavages)
# call the ProteaseDigestion::digest function
# which will return the number of discarded digestions products
# and fill the current_digest list with digested peptide sequences
digestor.digest(aaseq.fromString(fe.sequence), current_digest, options.min_length,
↳ options.max_length)
```

The next step is to use `FASTAFile` class to read the fasta input:

```
# construct a FASTAFile Object and read the input file
ff = FASTAFile()

ff.readStart(options.infile)

# construct and FASTAEntry Object
fe = FASTAEntry()

# loop over the entry in the fasta while using while
while(ff.readNext(fe)):
```

The output `idXML` needs the information about protein and peptide level, which can be saved in the `ProteinIdentification` and `PeptideIdentification` classes.

```
idxml = IdXMLFile()
idxml.store(options.outfile, protein_identifications, peptide_identifications)
```

This is the part of the program which unifies the snippets provided above. Please have a closer look how the protein and peptide datastructure is incorporated in the program.

```
def main(options):
    # read fasta file
    ff = FASTAFile()
    ff.readStart(options.infile)

    fe = FASTAEntry()

    # use ProteaseDigestion class
    digestor = ProteaseDigestion()

    digestor.setEnzyme(options.enzyme)
    digestor.setMissedCleavages(options.missed_cleavages)

    # protein and peptide datastructure
```

(continues on next page)

(continued from previous page)

```

protein_identifications = []

peptide_identifications = []
protein_identification = ProteinIdentification()

protein_identifications.append(protein_identification)
temp_pe = PeptideEvidence()

# number of dropped peptides due to length restriction
dropped_by_length = 0

while(ff.readNext(fe)):

    # construct ProteinHit and fill it with sequence information
    temp_protein_hit = ProteinHit()

    temp_protein_hit.setSequence(fe.sequence)
    temp_protein_hit.setAccession(fe.identifier)

    # save the ProteinHit in a ProteinIdentification Object

    protein_identification.insertHit(temp_protein_hit)

    # construct a PeptideHit and save the ProteinEvidence (Mapping) for the specific_
    ↪ current protein

    temp_peptide_hit = PeptideHit()
    temp_pe.setProteinAccession(fe.identifier);

    temp_peptide_hit.setPeptideEvidences([temp_pe])

    # digestion

    current_digest = []
    aaseq = AASequence()
    if (options.enzyme == "none"):

        current_digest.append(aaseq.fromString(fe.sequence))
    else:

        ↪ dropped_by_length += digester.digest(aaseq.fromString(fe.sequence), current_
        ↪ digest, options.min_length, options.max_length)

    for seq in current_digest:
        # fill the PeptideHit and PeptideIdentification datastructure

        peptide_identification = PeptideIdentification()

```

(continues on next page)

(continued from previous page)

```

temp_peptide_hit.setSequence(seq)

peptide_identification.insertHit(temp_peptide_hit)

peptide_identifications.append(peptide_identification)

print(str(dropped_by_length) + " peptides have been dropped due to the length_
↪restriction.")

idxml = IdXMLFile()
idxml.store(options.outfile, protein_identifications, peptide_identifications)

```

Putting things together

The parameter input and the functions can be used to construct the program we are looking for. If you are struggling please have a look in the example data section `ProteinDigestor.py`.

Now you can run your tool in the Anaconda Terminal `ExampleDatapyopenms`.

```

python ProteinDigestor.py -in mini_example.fasta -out mini_example_out.idXML -enzyme_
↪Trypsin -min_length 6 -max_length 40 -missed_cleavages 1

```

Bonus task

Task

Implement all other 184 TOPP tools using `pyOpenMS`.

1.15.11 Quality control

Introduction

In this chapter, we will build on an existing workflow with `OpenMS` / `KNIME` to add some quality control (QC). We will utilize the `qcML` tools in `OpenMS` to create a file with which we can collect different measures of quality to the mass spectrometry runs themselves and the applied analysis. The file also serves the means of visually reporting on the collected quality measures and later storage along the other analysis result files. We will, step-by-step, extend the label-free quantitation workflow from section 3 with QC functions and thereby enrich each time the report given by the `qcML` file. But first, to make sure you get the most of this tutorial section, a little primer on how we handle QC on the technical level.

QC metrics and qcML

To assert the quality of a measurement or analysis we use quality metrics. Metrics are describing a certain aspect of the measurement or analysis and can be anything from a single value, over a range of values to an image plot or other summary. Thus, qcML metric representation is divided into QC parameters (QP) and QC attachments (QA) to be able to represent all sorts of metrics on a technical level. A QP may (or may not) have a value which would equal a metric describable with a single value. If the metric is more complex and needs more than just a single value, the QP does not require the single value but rather depends on an attachment of values (QA) for full meaning. Such a QA holds the plot or the range of values in a table-like form. Like this, we can describe any metric by a QP and an optional QA. To assure a consensual meaning of the quality parameters and attachments, we created a controlled vocabulary (CV). Each entry in the CV describes a metric or part/extension thereof. We embed each parameter or attachment with one of these and by doing so, connect a meaning to the QP/QA. Like this, we later know exactly what we collected and the programs can find and connect the right dots for rendering the report or calculating new metrics automatically. You can find the constantly growing controlled vocabulary [here](#). Finally, in a qcml file, we split the metrics on a per mass-spectrometry-run base or a set of mass-spectrometry-runs respectively. Each run or set will contain its QP/QA we calculate for it, describing their quality.

Building a qcML file per run

As a start, we will build a basic qcML file for each mzML file in the label-free analysis. We are already creating the two necessary analysis files to build a basic qcML file upon each mzML file, a feature file and an identification file. We use the **QCCalculator** node from **Community > OpenMS > Utilities** where also all other QC* nodes will be found. The **QCCalculator** will create a very basic qcML file in which it will store collected and calculated quality data.

- Copy your label-free quantitation workflow into a new lfq-qc workflow and open it.
- Place the **QCCalculator** node after the **IDMapper** node. Being inside the **ZipLoop**, it will execute for each of the three mzML files the **Input** node.
- Connect the first **QCCalculator** port to the first **ZipLoopStart** outlet port, which will carry the individual mzML files.
- Connect the last's ID outlet port (**IDFilter** or the ID metanode) to the second **QCCalculator** port for the identification file.
- Finally, connect the **IDMapper** outlet to the third **QCCalculator** port for the feature file.

The created qcML files will not have much to show for, basic as they are. So we will extend them with some basic plots.

- First, we will add an 2D overview image of the given mass spectrometry run as you may know it from TOPPView. Add the **ImageCreator** node from **Community Nodes > OpenMS > Utilities**. Change the width and height parameters to 640x640 as we don't want it to be too big. Connect it to the first **ZipLoopStart** outlet port, so it will create an image file of the mzML's contained run.
- Now we have to embed this file into the qcML file, and attach it to the right **QualityParameter**. For this, place a **QCEmbedder** node behind the **ImageCreator** and connect that to its third inlet port. Connect its first inlet port to the outlet of the **QCCalculator** node to pass on the qcML file. Now change the parameter `cv_acc` to QC:0000055 which designates the attached image to be of type QC:0000055 - MS experiment heatmap. Finally, change the parameter `qp_att_acc` to QC:0000004, to attach the image to the QualityParameter QC:0000004 - MS acquisition result details.
- For a reference of which CVs are already defined for qcML, have a look at the following [link](#).

There are two other basic plots which we almost always might want to look at before judging the quality of a mass spectrometry run and its identifications: the **total ion current** (TIC) and the **PSM mass error** (Mass accuracy), which we have available as pre-packaged QC metanodes.

Task

Import the workflow from WorkflowsQuality ControlQC Metanodes.zip by navigating to **File > Import KNIME Workflow...**

- Copy the **Mass accuracy** metanode into the workflow behind the **QCEmbedder** node and connect it. The qcML will be passed on and the Mass accuracy plots added. The information needed was already collected by the **QCCalculator**.
- Do the same with the **TIC** metanode so that your qcML file will get passed on and enriched on each step.

R Dependencies: This section requires that the R packages **ggplot2** and **scales** are both installed. This is the same procedure as in this section. In case that you use an R installation where one or both of them are not yet installed, open the **R Snippet** nodes inside the metanodes you just used (double-click). Edit the script in the *R Script* text editor from:

```
#install.packages("ggplot2")
#install.packages("scales")
```

to

```
install.packages("ggplot2")
install.packages("scales")
```

Press **Eval script** to execute the script.

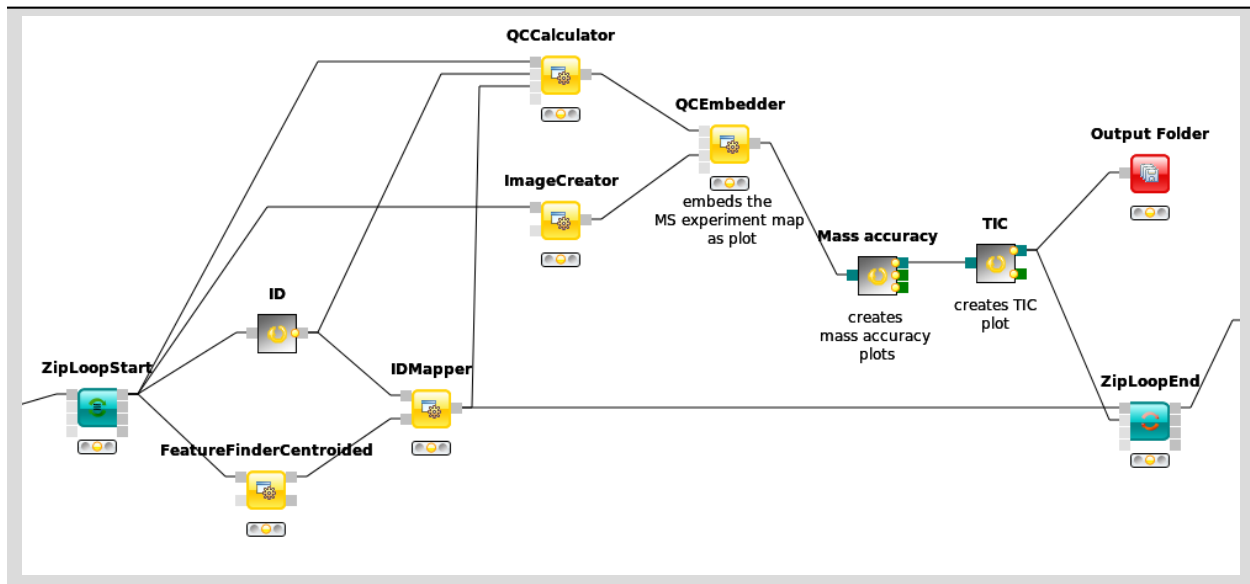


Figure 51: Basic QC setup within a LFQ workflow.

Note: To have a peek into what our qcML now looks like for one of the **ZipLoop** iterations, we can add an **Output Folder** node from **Community Nodes > GenericKnimeNodes > IO** and set its destination parameter to somewhere we want to find our intermediate qcML files in, for example **tmp > qcxfq**. If we now connect the last metanode with the Output Folder and restart the workflow, we can start inspecting the qcML files.

Task

Find your first created qcML file and open it with the browser (not IE), and the contained QC parameters will be rendered for you.

Adding brand new QC metrics

We can also add brand new QC metrics to our qcML files. Remember the **Histogram** you added inside the **ZipLoop** during the label-free quantitation section? Let's imagine for a moment this was a brand new and utterly important metric and plot for the assessment of your analyses quality. There is an easy way to integrate such new metrics into your qcMLs. Though the **Histogram** node cannot pass its plot to an image, we can do so with a **R View (table)**.

- Add an **R View (table)** next to the **IDTextReader** node and connect them.
- Edit the **R View (table)** by adding the *R Script* according to this:

```
#install.packages("ggplot2")
library("ggplot2")
ggplot(knime.in, aes(x=peptide_charge)) +

geom_histogram(binwidth=1, origin =-0.5) +
scale_x_discrete() +

ggtitle("Identified peptides charge histogram") +
ylab("Count")
```

- This will create a plot like the **Histogram** node on *peptide_charge* and pass it on as an *image*.
- Now add and connect a **Image2FilePort** node from **Community Nodes > GenericKnimeNodes > Flow** to the **R View (table)**.
- We can now use a **QCEmbedder** node like before to add our new metric plot into the qcML.
- After looking for an appropriate target from the following [link](#), we found that we can attach our plot to the *MS identification result details* by setting the parameter *qp_att_acc* to QC:0000025, as we are plotting the charge histogram of our identified peptides.
- To have the plot later displayed properly, we assign it the parameter *cv_acc* of QC:0000051, a generic plot. Also we made sure in the *R Script*, that our plot carries a caption so that we know which is which, if we had more than one new plot.
- Now we redirect the **QCEmbedders** output to the **Output Folder** from before and can have a look at how our qcML is coming along after restarting the workflow.

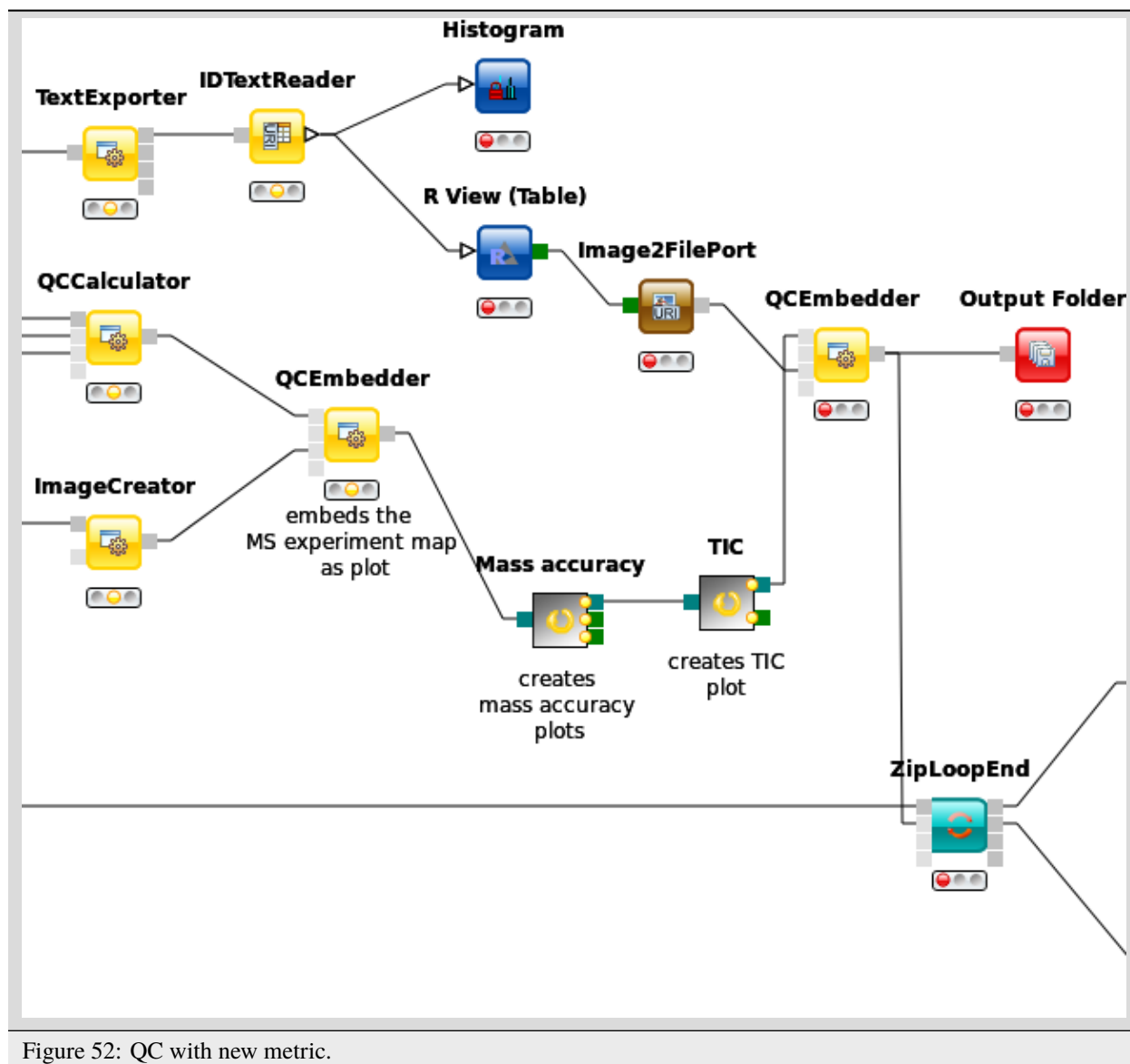


Figure 52: QC with new metric.

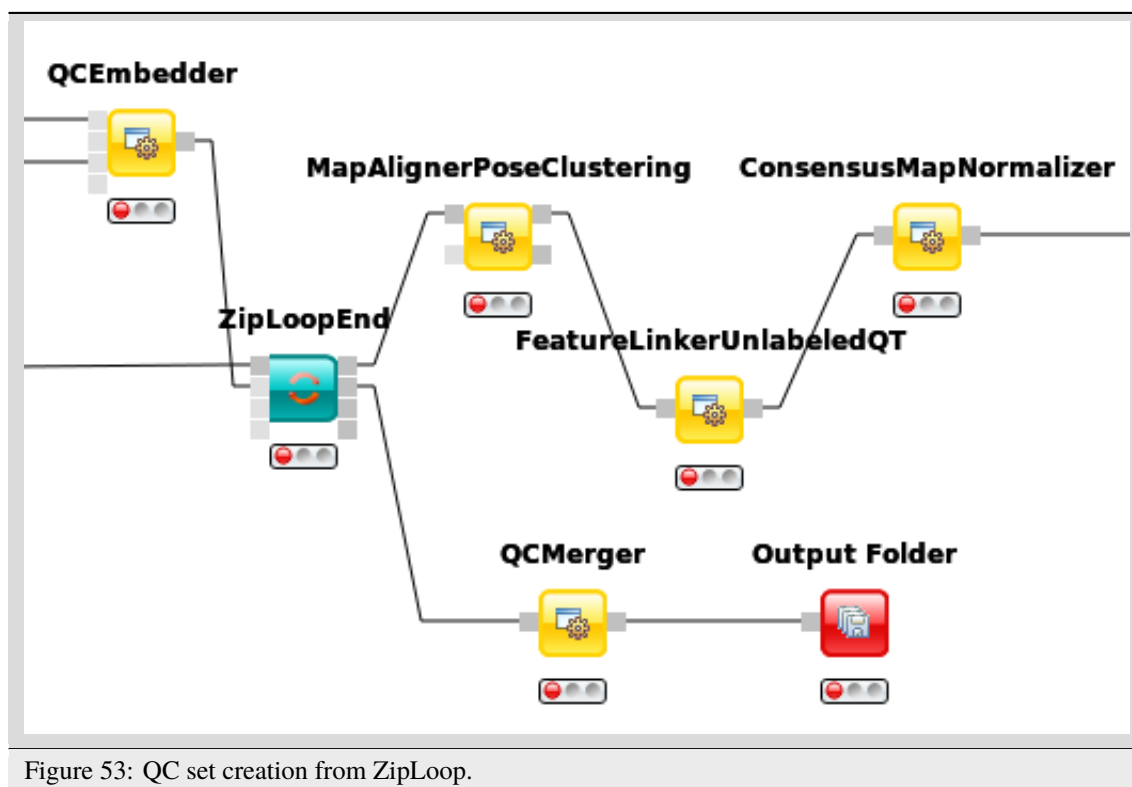
Set QC metrics

Besides monitoring the quality of each individual mass spectrometry run analysis, another capability of QC with OpenMS and qcML is to monitor the complete set. The easiest control is to compare mass spectrometry runs which should be similar, e.g. technical replicates, to spot any aberrations in the set. For this, we will first collect all created qcML files, merge them together and use the qcML onboard set QC properties to detect any outliers.

- Connect the **QCEmbedders** output from last section to the **ZipLoopEnds** second input port.
- The corresponding output port will collect all qcML files from each **ZipLoop** iteration and pass them on as a list of files.
- Now we add a **QCMerger** node after the **ZipLoopEnd** and feed it that list of qcML files. In addition, we set its parameter `setname` to give our newly created set a name - say `spikein_replicates`.
- To inspect all the QCs next to each other in that created qcML file, we have to add a new **Output Folder** to which

we can connect the **QCMerger** output.

When inspecting the set-qcML file in a browser, we will be presented another overview. After the set content listing, the basic QC parameters (like number of identifications) are each displayed in a graph. Each set member (or run) has its own section on the x-axis and each run is connected with that graph via a link in the mouseover on one of the QC parameter values.



Task

For ideas on new QC metrics and parameters, as you add them in your qcML files as generic parameters, feel free to [contact us](#), so we can include them in the CV.

1.15.12 Troubleshooting guide

This section will show you where you can turn to when you encounter any problems with this tutorial or with our nodes in general. Please see the [FAQ](#) first. If your problem is not listed or the proposed solution does not work, feel free to leave us a message at the means of support that you see most fit. If that is the case, please provide us with as much information as you can. In an ideal case, that would be:

- Your operating system and its version (e.g. Windows 8, Ubuntu 14.04).
- Your KNIME version (e.g. KNIME 3.1.2 full, KNIME 3.1.1 core).
- If not full: Which update site did you use for the OpenMS plugin? Trunk (nightly-builds) or Stable?
- Your OpenMS plugin version found under **Help > Install New Software > What is already installed?**
- Other installations of OpenMS on your computer (e.g. from the independent OpenMS installer, another KNIME instance etc.)

- The log of the error in KNIME and the standard output of the tool (see FAQ: How to debug).
- Your description of what you tried to do and experienced instead.

FAQ

How to debug KNIME and/or the OpenMS nodes?

- **KNIME:** Start with the normal log on the bottom right of KNIME. In general all warnings and errors will be listed there. If the output is not helpful enough, try to set the logging verbosity to the highest (DEBUG) under **Preferences > KNIME > Log file log level**.
- **OpenMS nodes:** The first step should also be the log of KNIME. Additionally, you can view the output and the errors of our tools by right-clicking on the node and selecting **View: NODENAME Std Output?error**. This shows you the output of the OpenMS executable that was called by that node. For advanced users, you can try to execute the underlying executable in your `KNIME/plugins/de.openms.platform.arch.version/payload/bin` folder, to see if the error is reproducible outside of KNIME. You can look up temporary files that are created by OpenMS nodes not connected to an Output or Viewer Node by right-clicking on a node and selecting the corresponding output view for the output you want to have a look at. The output views are located on the bottom of the menu that shows up after right-clicking. Their icon is a magnifying glass on top of a data table. The names of the output views in that menu may vary from node to node (usually a combination of "file", "out", "output" and optionally its possible extensions). For example for the Input File node you can open the information on the output files by clicking on "loaded file". In any case, a hierarchy of file descriptions will show up. If there are multiple files on that port they will be numbered (usually beginning from 0). Expand the information for the file you want to see and copy its URI (you might need to erase the "file:" prefix). Now open it with an editor of your choice. Be aware that temporary files are subject to deletion and are usually only stored as long as they are actually needed. There is also a Debug mode for the GKN nodes that keeps temporary files that can be activated under **Preferences > KNIME > Generic KNIME Nodes > Debug mode**. For the single nodes you can also increase the debug level in the configuration dialog under the advanced parameters. You can also specify a log file there, to save the log output of a specific node on your file system.

General

Q: Can I add my own modifications to the Unimod.xml?

A: Unfortunately not very easy. This is an open issue since the selections are hard-coded during creation of the tools. We included 10 places for dummy modifications that can be entered in our Unimod.xml and selected in KNIME.

Q: I have problem XYZ but it also occurs with other nodes or generally in the KNIME environment/GUI, what should I do?

A: This sounds like a general KNIME bug and we advise to search help directly at the KNIME developers. They also provide a [FAQ](#) and a [forum](#).

Q: After exporting and reading in results into a KNIME table (e.g. with a MzTabExporter and MzTabReader combination) numeric values get rounded (e.g. from scientific notation 4.5e-10 to zero) or are in a different representation than in the underlying exported file!

A: Please try a different table column renderer in KNIME. Open the table in question, right-click on the header of an affected column and select another Available Renderer by hovering and finally left-clicking.

Q: I have checked all the configurations but KNIME complains that it can not find certain output Files (FileStoreObjects).

A: Sometimes KNIME/GKN has hiccups with multiple nodes with a same name, executed at the same time in the same loop. We have seen that a simple save and restart of KNIME usually solves the problem.

Platform-specific problems

Linux

Q: Whenever I try to execute an OpenMS node I get an error similar to these:

```
/usr/lib/x86_64-linux-gnu/libgomp.so.1: version `GOMP_4.0' not found  
/usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3.4.20' not found
```

A: We currently build the binaries shipped in the OpenMS KNIME plugin with gcc 4.8. We will try to extend our support for older compilers. Until then you either need to upgrade your gcc compiler or at least the library that the tool complained about or you need to build the binaries yourself (see OpenMS documentation) and replace them in your KNIME binary folder (YOURKNIMEFOLDER/plugins/de.openms.platform.architecture.version/payload/bin)

Q: Why is my configuration dialog closing right away when I double-click or try to configure it? Or why is my GUI responding so slow?

A: If you have any problems with the KNIME GUI or the opening of dialogues under Linux you might be affected by a GTK bug. See the KNIME forum (e.g. [here](#) or [here](#)) for a discussion and a possible solution. In short: set environment variable by calling `export SWT_GTK3=0` or edit `knime.ini` to make Eclipse use GTK2 by adding the following two lines:

```
-launcher.GTK_version  
2
```

macOS

Q: I have problems installing RServe in my local R installation for the R KNIME Extension.

A: If you encounter linker errors while running `install.packages("Rserve")` when using an R installation from homebrew, make sure gettext is installed via homebrew and you pass flags to its lib directory. See [StackOverflow question 21370363](#).

Q: Although Ctrl + Left-click TOPPAS.app or TOPPView.app and accept the risk of a downloaded application, the icon only shortly blinks and nothing happens.

A: It seems like your OS is not able to remove the quarantine flag. If you trust us, please remove it yourself by typing the following command in your Terminal.app:

```
xattr -r -d {{ 'com.apple.quarantine /Applications/OpenMS-{{0}}'.format(version) }}
```

Windows

Q: KNIME has problems getting the requirements for some of the OpenMS nodes on Windows, what can I do?

A: Get the prerequisites installer [here](#) or install .NET3.5, .NET4 and VCRedist10.0 and 12.0 yourself.

Nodes

Q: Why is my XTandemAdapter printing empty or VERY few results, although I did not use an e-value cutoff?

A: Due to a bug in OpenMS 2.0.1, the XTandemAdapter requires a default parameter file. Give it the default configuration in `YOURKNIMEFOLDER/plugins/de.openms.platform.architecture.version/payload/share/CHEMISTRY/XTandemdefaultinput.xml` as a third input file. This should be resolved in newer versions though, such that it automatically uses this file if the optional inputs is empty. This should be solved in newer versions.

Q: Do MSGFPlusAdapter, LuciphorAdapter or SiriusAdapter generally behave different/unexpected?

A: These are Java processes that are started underneath. For example they can not be killed during cancellation of the node. This should not affect its performance, however. Make sure you set the Java memory parameter in these nodes to a reasonable value. Also MSGFPlus is creating several auxiliary files and accesses them during execution. Some users therefore experienced problems when executing several instances at the same time.

Sources of support

If your questions could not be answered by the FAQ, please feel free to turn to our developers via one of the following means:

- File an issue on [GitHub](#)
- Write to the [Mailing List](#)
- Open a thread on the KNIME Community Contributions [forum](#) for OpenMS

1.15.13 References

1.16 Worked Examples: Different OpenMS Methods to Achieve the Same Outcome

The following tutorials provide the opportunity to complete a real-world example while also seeing the different methods OpenMS makes available to complete the same task.

There are four ways to use OpenMS to complete a task. For example, say you want to read a file and store the information in an output file. You can do this by:

- **Using TOPP shell** to run a shell script or execute a command directly.
- **Using pyOpenMS** and creating and running a python script.
- **Using TOPPView**, a graphical user interface provided by OpenMS.
- **Constructing a workflow in KNIME**, which can be saved and executed on multiple input files.
- **Running a script** using Nextflow, a language based on the Groovy programming language.

The following sections explain how to read a file and store the information in an output file using these five different methods.

1.16.1 Using TOPP shell

As explained in the command-line quick start guide, TOPP shell is available for those who are comfortable with executing command line tools and writing scripts.

To read the information of a file, just type the following in the command line and press Enter.

```
FileInfo -in <insert input file> -out <insert output file>
```

You can also copy and paste this into a shell script (file with an .sh extension), and then type bash in the command line to execute the script.

1.16.2 Using pyOpenMS

You can replicate the functionality of the FileInfo TOPP tool in pyOpenMS, using one of the following examples, depending on the type of file you want to get information about.

- Example for mzML file

```
from pyopenms import *

exp = MSExperiment()
MzMLFile().load("sample.mzML", exp)
exp.updateRanges()

ms_levels = exp.getMSLevels()
num_spectra = {level: 0 for level in ms_levels}

for spec in exp:
    num_spectra[spec.getMSLevel()] += 1

print("Instrument:")
for analyzer in exp.getInstrument().getMassAnalyzers():
    print(f"\tMass Analyzer: {analyzer.getType()} (resolution: {analyzer.
    ↳getResolution()})")

print("\nMS levels: "+, ".join([str(level) for level in ms_levels]))
print(f"Total number of peaks: {sum([spec.size() for spec in exp])}")
print(f"Total number of spectra: {exp.size()}")

print("\nRanges:")
print(f"\tretention time: {exp.getMinRT()} .. {exp.getMaxRT()} ({round((exp.
    ↳getMaxRT()-exp.getMinRT())/60, 2)} min)")
print(f"\tmass-to-charge: {exp.getMinMZ()} .. {exp.getMaxMZ()}")
print(f"\tintensity: {exp.getMinIntensity()} .. {exp.getMaxIntensity()}")

print("\nNumber of spectra per MS level:")
for level, number in num_spectra.items():
    print(f"\tlevel {level}: {number}")
```

- Example for featureXML file

```
feature_map = FeatureMap()
FeatureXMLFile().load("sample.featureXML", feature_map)
```

(continues on next page)

(continued from previous page)

```

charges = {}
number_of_ids = {}
tic = 0

for feature in feature_map:
    charge = feature.getCharge()
    if charge in charges.keys():
        charges[charge] += 1
    else:
        charges[charge] = 1
    num_ids = len(feature.getPeptideIdentifications())
    if num_ids in number_of_ids.keys():
        number_of_ids[num_ids] += 1
    else:
        number_of_ids[num_ids] = 1
    tic += feature.getIntensity()

print(f"Number of features: {feature_map.size()}")

print("\nRanges:")
print(f"\tretention time: {feature_map.getMinRT()} .. {feature_map.getMaxRT()} (
↳{round((feature_map.getMaxRT()-feature_map.getMinRT())/60, 2)} min)")
print(f"\tmass-to-charge: {feature_map.getMinMZ()} .. {feature_map.getMaxMZ()}")
print(f"\tintensity: {feature_map.getMinIntensity()} .. {feature_map.
↳getMaxIntensity()}")

print(f"\nTotal ion current in features: {int(tic)}")

print("\nCharge distribution:")
for charge, occurence in charges.items():
    print(f"\tcharge {charge}: {occurence}x")

print("\nDistribution of peptide identifications (IDs) per feature:")
for num_ids, occurence in number_of_ids.items():
    print(f"\t{num_ids} IDs: {occurence}")

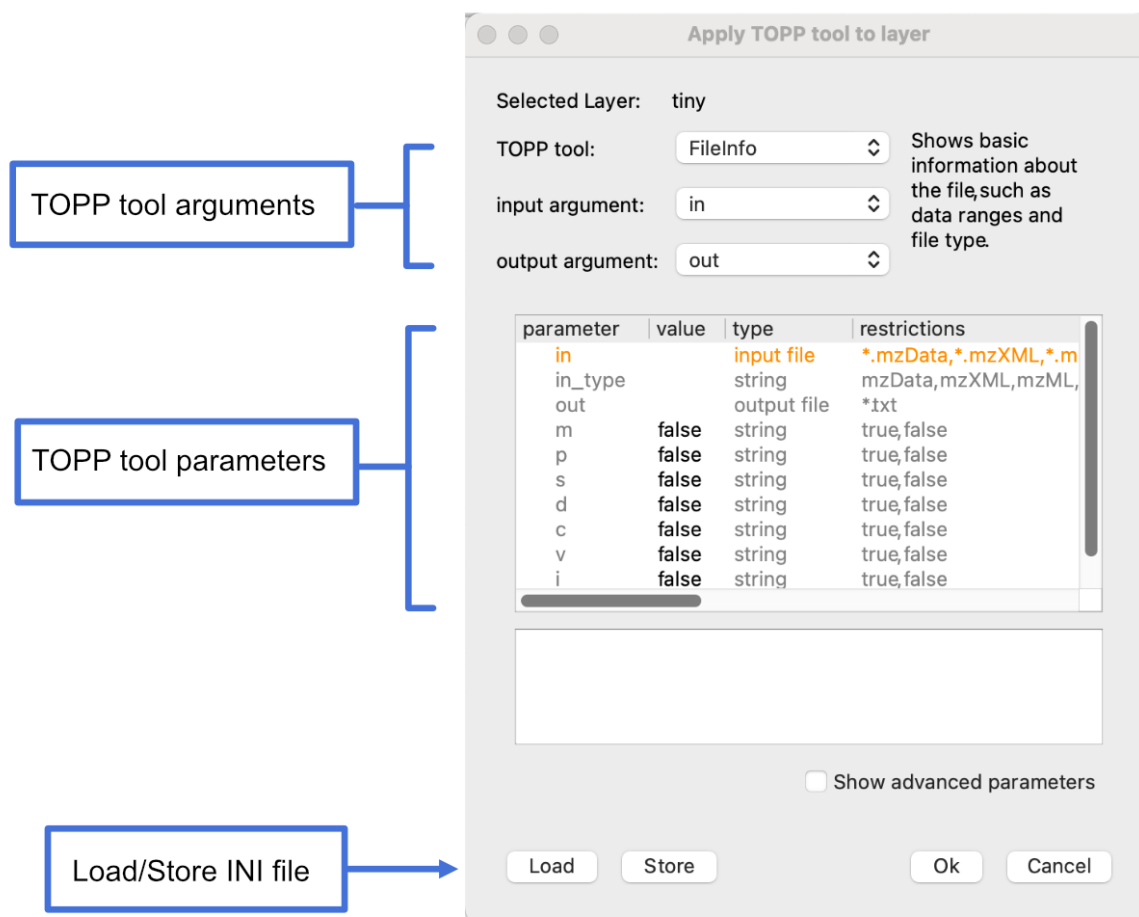
print(f"\nUnassigned peptide identifications: {len(feature_map.
↳getUnassignedPeptideIdentifications())}")

```

1.16.3 Using TOPPView

If you want a graphical user interface to interact with, then use TOPPView. Follow these steps to read the file information using TOPPView:

1. Go to **File > Open file** and open a file by following the prompts.
2. Go to **Tools > Apply TOPP tool (whole layer)**.
3. Set **TOPP tool** to **FileInfo** and **output argument** to **out**.
4. Load an existing INI file by clicking **Load** and selecting an INI file from the file importer. If you don't have an INI file, click **Store**, enter a file name and click **OK** to generate an INI file, then click **Load** and load the file.



5. Click **OK**.
6. Open the **Log** panel at the bottom of the screen to view the resulting file information.

1.16.4 Constructing a workflow in KNIME

KNIME is available for those who want a graphical user interface to create and use workflows. Here is an example of how to report file information on an input file to an output file using KNIME.

1. **Install OpenMS plugin.**
2. **Open a new file** by going to **File > New file**.
3. **Construct workflow by adding nodes.**
4. **Configure the nodes in workflow.**
5. **Play the workflow.**

1.16.5 Running a Nextflow script

Nextflow is a scripting language based on the Groovy programming language.

The following is a Nextflow script that executes the FileInfo TOPP tool on an input file.

```
params.input_file = 'path/to/input_file'

process runFileInfo {
  output:
    stdout

  """
  FileInfo -in ${params.input_file}
  """
}

workflow {
  runFileInfo | view { it }
}
```

The above script can be executed from the command line by entering:

```
nextflow run file_info.nf
```

You can also specify a file path from the command line by using:

```
nextflow run file_info.nf --input_file path/to/file
```

1.17 OpenMS C++ Core Library

This page is under construction.

1.18 Build From Source

1.18.1 GNU Linux

To build OpenMS from source on a GNU Linux machine, follow the build instructions for [Linux](#).

Dockerfiles to build different kind of images (corresponding to build instructions, e.g. on ArchLinux) can be found on GitHub in [OpenMS/dockerfiles](#) repository.

1.18.2 macOS

To build OpenMS from source on a macOS machine, follow the build instructions for [macOS](#).

1.18.3 Windows

To build OpenMS from source on a Windows machine, follow the build instructions for [Windows](#).

1.19 External Code using OpenMS

If OpenMS' TOPP and UTILS tools are not enough in a certain scenario, you can either request a change to OpenMS, if you feel this functionality is useful for others as well, or modify/extend OpenMS privately. For the latter, there are multiple ways to do this:

- Modify the developer version of OpenMS by changing existing tools or adding new ones.
- Use an **External Project** to write a new tool, while not touching OpenMS itself (see below on how to do that).

Once you've finished your new tool, and it only needs to run on the development machine. To ship it to a new client machine, see, read further in this document.

1.19.1 Compiling external code

It is very easy to set up an environment to write your own programs using OpenMS. Make sure to download and installed the source package of OpenMS/TOPP properly.

Note: You cannot use the `install` target when working with the development version of OpenMS, it must be built and used within the build tree.

All important compiler settings and preprocessor definitions along with the OpenMS library are available. The most important variables are:

- `OpenMS_INCLUDE_DIRECTORIES`: all include directories containing OpenMS headers
- `OPENMS_ADDCXX_FLAGS`: preprocessor macros we require written as `(-DMACRO1 -DMACRO2)`

and the OpenMS target itself (which you can link against).

The example that follows will be explained in details:

```
### example CMakeLists.txt to develop C++ programs using OpenMS
project("Example_Project_using_OpenMS")
cmake_minimum_required(VERSION 3.0)

## list all your executables here (a corresponding .cpp file should exist, e.g. Main.cpp)
set(my_executables
  Main
)

## list all classes here, which are required by your executables
## (all these classes will be linked into a library)
set(my_sources
```

(continues on next page)

(continued from previous page)

```

ExampleLibraryFile.cpp
)

## find OpenMS configuration and register target "OpenMS" (our library)
find_package(OpenMS)
## if the above fails you can try calling cmake with -D OpenMS_DIR=/path/to/OpenMS/
## or modify the find_package() call accordingly
## find_package(OpenMS PATHS "</path/to/OpenMS//")

# check whether the OpenMS package was found
if (OpenMS_FOUND)
    message(STATUS "\nFound OpenMS at ${OpenMS_DIR}\n")

    ## library with additional classes from above
    add_library(my_custom_lib STATIC ${my_sources})

    ## add targets for the executables
    foreach(i ${my_executables})
        add_executable(${i} ${i}.cpp)
        ## link executables against OpenMS
        target_link_libraries(${i} OpenMS my_custom_lib)
    endforeach(i)

else(OpenMS_FOUND)
    message(FATAL_ERROR "OpenMSConfig.cmake file not found!")
endif(OpenMS_FOUND)

```

The command `project` defines the name of the project, the name is only of interest if you're working in an IDE or want to export this project's targets. To compile the program, append it to the `my_executables` list. If you use object files (classes which do not contain a main program), append them to the `my_sources` list. In the next step CMake creates a statically linked library of the object files, listed in `my_sources`. This simple `CMakeLists.txt` example can be extended to also build shared libraries, include other external libraries and so on.

An example external project can be found in `OpenMS/share/OpenMS/examples/external_code`. Copy these files to a separate directory and use CMake to configure it (here as an in-source build).

```

cd <path_to_external_project>
cmake -G "<generator>" .

```

For more information visit the website of cmake at cmake.org and consult the documentation.

Important: Have fun coding with OpenMS!

1.19.2 Shipping external code to a new machine

If you've modified OpenMS itself and not used an external project use our installer scripts, to build your own OpenMS installer for your platform (see our internal FAQ which is built using "make doc_internal") and ship that to a client machine.

If you've used an external project and have a new executable (+ an optional new library), use the installer approach as well, and manually copy the new executable to the TOPP/UTILS binary directory (e.g. on Windows this could be `c:/program files/OpenMS/bin`, on Linux it could be `/bin`).

If you do not use the installer, copy all required files manually, plus a few extra steps, see below. What needs to be done is a little platform dependent, thus very cumbersome to explain. Look at the cmake installer scripts, to see whats required (for macOS and Linux see `OpenMS/cmake/package*.cmake`).

In short:

- copy the `OpenMS/share/OpenMS` directory to the client machine (e.g `<client/my_dir>/share`) and set the environment variable `OPENMS_DATA_PATH` to this directory
- copy the OpenMS library (`OpenMS.dll` for Windows or `OpenMS.so/.dylib` for Linux/macOS) to `<client/my_dir>/bin`.
- copy all Qt4 libraries to the client `<client/my_dir>/bin` or on Linux/macOS make sure you have installed the Qt4 package.
- [Windows only] copy Xerces dll (see `contrib/lib`) to `<client/my_dir>/bin`
- [Windows only] install the VS redistributable package (see Microsoft Homepage) on the client machine which corresponds to the VS version that was used to compile your code (use the correct redistributable package!, i.e., architecture 32|64bit, VS version, VS Service Pack version). If you choose the wrong redistributable package, you will get "Application failed to initialize properly..." error messages.

1.20 Developer Tutorial

This page is under construction.

1.21 Contributor's Quick Start Guide

To contribute to OpenMS:

- Familiarise yourself with the *OpenMS online documentation*.
- Learn how to *build OpenMS*.
- Check out the *OpenMS tutorial for developers*.

For any questions, please *contact us*.

1.21.1 Technical documentation

Note: Untested installers and containers are known as the *nightly snapshot*, are released every night. They generally pass automated continuous integration tests but no manual tests.

View the documentation for the nightly snapshot of OpenMS [develop branch](#) at the [build archive](#).

See the documentation for the [latest release](#).

1.21.2 Contribution guidelines

Before contributing to OpenMS, read information on the development model and conventions followed to maintain a coherent code base.

Development model

OpenMS follows the [Gitflow development workflow](#).

Every contributor is encouraged to create their own fork (even if they are eligible to push directly to OpenMS). To create a fork:

1. Follow the documentation on [forking](#).
2. Keep your fork [up-to-date](#).
3. Create a [pull request](#). Before opening the pull request, please view the [pull request guidelines](#).

Coding conventions

See the manual for coding style recommended by OpenMS: [Coding conventions](#).

See also:

[C++ Guide](#).

OpenMS automatically tests for common coding convention violations using a modified version of `cpplint`. Style testing can be enabled using `cmake` options. `clang-format` is used for formatting the `cpp` code.

Commit messages

View the guidelines for commit messages: [How to write commit messages](#).

Automated unit tests

Nightly tests run on different platforms. It is recommended to test on different platforms.

Tip: This saves time and increases productivity during continuous integration tests.

Nightly tests: [CDASH](#).

1.21.3 Further contributor resources

Consider the following resources for further information:

- **Guidelines for adding new dependency libraries:** View the guidelines for [adding new dependency libraries](#).
- **Experimental installers:** We automatically build installers for different platforms. These usually contain unstable or partially untested code. The nightly (unstable) installers are available at the [build archive](#).
- **Developer FAQ:** Visit the [Developer FAQ](#) to get answers to frequently asked questions.

1.22 OpenMS Git Workflow

Before getting started, install latest version of git to avoid problems like GitHub https authentication errors (see [Troubleshooting cloning errors](#) and a solution using [ssh](#)).

OpenMS follows the [git flow workflow](#). The difference is that merge commits are managed via pull requests instead of creating merge commits locally.

1.22.1 Naming conventions

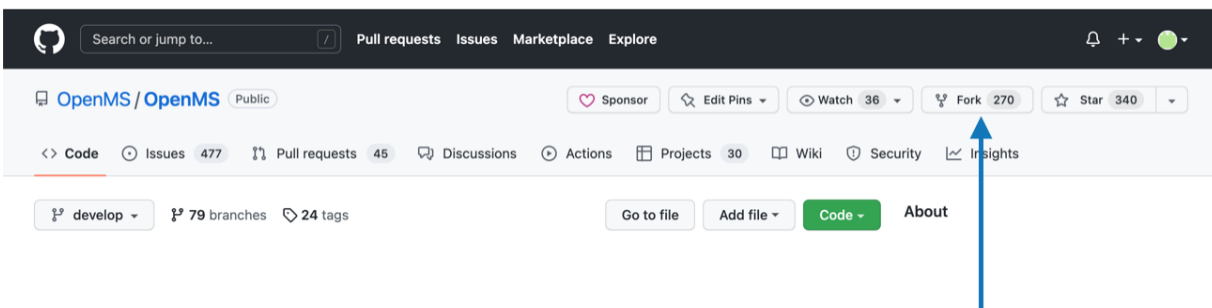
Naming conventions for the following apply:

- A **local repository** is the repository that lies on your hard drive after cloning.
- A **remote repository** is a repository on a git server such as GitHub.
- A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.
- **Origin** refers to a remote repository that you have forked. Call this repository `https://github.com/_YOURUSERNAME_/OpenMS`.
- **Upstream** refers to the original remote OpenMS repository. Call this repository `https://github.com/OpenMS/OpenMS`.

1.22.2 Create fork

Start by [forking](#) the OpenMS repository.

To create a fork, click **Fork** under the main menu as shown below.



1.22.3 Clone your fork

To obtain a local repository copy, clone your fork using:

```
$ git clone https://github.com/_YOURUSERNAME_/OpenMS.git
```

This will clone your fork (correctly labelled `origin` by default) into a local copy on your computer.

Note: To use `git clone git@github.com:_YOURUSERNAME_/OpenMS.git`, make sure you have [SSH key added to your GitHub account](#).

1.22.4 Link remote branches to your local working repository

After cloning your fork, your local repository should be named `origin`. Validate this by executing:

```
$ git remote -v
origin https://github.com/_YOURUSERNAME_/OpenMS.git (fetch)
origin https://github.com/_YOURUSERNAME_/OpenMS.git (push)
```

Sync data between your local copy, your fork (`origin`) and the remote original OpenMS/OpenMS repository (`upstream`) by using the following command:

```
$ git remote add upstream https://github.com/OpenMS/OpenMS.git
```

Verify that upstream was added correctly by calling:

```
$ git remote -v
origin https://github.com/_YOURUSERNAME_/OpenMS.git (fetch)
origin https://github.com/_YOURUSERNAME_/OpenMS.git (push)
upstream https://github.com/OpenMS/OpenMS.git (fetch)
upstream https://github.com/OpenMS/OpenMS.git (push)
```

Fetch changes and new branches from your fork (`origin`) as well as from the central, upstream OpenMS repository by executing:

```
$ git fetch upstream
$ git fetch origin
```

or

```
$ git fetch --all
```

Create a local branch using the following:

```
$ git checkout -b <insert branch-name>
```

Call `git branch -va` to display the status of local and remote branches. You should see an output that looks like this:

```
$ git branch -va
* develop          349ec48 Merge pull request #691 from cbielow/MGF_fix
  feature/my_shiny_new_feature 3c05538 [FEATURE] added option to keep, ensure or
↪reassign UIDs during conversion
```

(continues on next page)

(continued from previous page)

```

remotes/origin/SILACAnalyzer      3ceae38 Fixed test.
remotes/origin/antiloop           3fe5aa3 git-svn-id: https://open-ms.svn.sourceforge.
↪net/svnroot/open-ms/branches/antiloop@12117 6adb6e08-d915-0410-941f-83917bcadc18
remotes/origin/develop            349ec48 Merge pull request #691 from cbieLOW/MGF_fix
remotes/origin/master             b182ba5 [NOP] first commit after SVN import to git
remotes/origin/msnovogen          93a5e4c [OPT] For faster access to specific amino
↪acids a ResidueServer was added.
remotes/upstream/HEAD             -> upstream/develop
remotes/upstream/SILACAnalyzer    3ceae38 Fixed test.
remotes/upstream/antiloop         3fe5aa3 git-svn-id: https://open-ms.svn.sourceforge.
↪net/svnroot/open-ms/branches/antiloop@12117 6adb6e08-d915-0410-941f-83917bcadc18
remotes/upstream/develop          349ec48 Merge pull request #691 from cbieLOW/MGF_fix
remotes/upstream/master           b182ba5 [NOP] first commit after SVN import to git
remotes/upstream/msnovogen        93a5e4c [OPT] For faster access to specific amino
↪acids a ResidueServer was added.

```

1.22.5 Keep your fork in sync

Keep your fork (origin) in sync with the OpenMS repository (upstream) by following the [GitHub instructions](#). In summary, to keep your fork in sync:

1. Fetch changes from upstream and update your local branch.
2. Push your updated local branch to your fork (origin).

Tip: To keep track of others repositories, use `git fetch --all --prune` to update them as well. The option `--prune` tells git to automatically remove tracked branches if they got removed in the remote repository.

```

$ git fetch --all --prune
$ git checkout develop
$ git merge --ff-only upstream/develop
$ git push origin develop

```

Feel free to experiment within your fork. However, for your code needs to meet OpenMS quality standards to be merged into the OpenMS repository,

Follow these rules:

- Never commit directly to the `develop` or `master` branches as it will complicate the merge.
- Try to start every feature from `develop` and not base features on other features.
- Name the OpenMS remote `upstream` and always push directly to `origin` (`git push origin <branch-name>`).
- When updating your fork, consider using `git fetch upstream` followed by `git merge --ff-only upstream/develop` to avoid creating merge commits in `develop`.
- If you never commit to `develop` this should always succeed and (if a commit accidentally went to `develop`) warn you instead of creating a merge commit.

1.22.6 Create new feature

All features start from `develop`.

```
$ git checkout develop
$ git checkout -b feature/your-cool-new-feature
```

All commits related to this feature will then go into the branch `feature/your-cool-new-feature`.

1.22.7 Keeping your feature branch in sync with develop branch

While working on your feature branch, it is usual that development continues and new features get integrated into the main development branch. This means your feature branch lags behind `develop`. To get your feature branch up-to-date, rebase your feature branch on `develop` using:

```
$ git checkout feature/myfeaturebranch
$ git rebase develop
```

The above commands:

1. Performs a rewind of your commits until the branching point.
2. Applies all commits that have been integrated into `develop`.
3. Reapplies your commits on top of the commits integrated into `develop`.

For more information, refer to a [visual explanation of rebasing](#).

Tip: Do not rebase published branches (e.g. branches that are part of a pull request). If you created a pull request, you should only add commits in your feature branch to fix things that have been discussed. After your pull request contains all fixes, you are ready to merge the pull request into `develop` without rebasing (see e.g. [rebase-vs-merge](#)).

1.22.8 Adding a feature to OpenMS

Features that should go into the main development line of OpenMS should be integrated via a [pull request](#). This allows the development community of OpenMS to discuss the changes and suggest possible improvements.

After opening the pull request via the GitHub web site, GitHub will try to create the pull request against the branch that you branched off from. Please check the branch that you are opening the pull request against before submitting the pull request. If any changes are made, a new pull request is required. Select **Allow others to make changes to this pull request** so that maintainers can directly help to solve problems.

Open pull requests only after checking code-style, documentation and passing tests. Pull requests that do not pass CI or code review will not be merged until the problems are solved. It is recommended that you read the [pull request guidelines](#) before you submit a pull request.

1.22.9 Update git submodules

Start in your local OpenMS/OpenMS repository (on your feature/pull request branch).

The following example uses a submodule called THIRDPARTY.

```
$ git submodule update --init THIRDPARTY
$ cd THIRDPARTY
# yes, in the submodules the default remote is origin
# usually you want to pull the changes from master (e.g. after your pull request to
  ↳ OpenMS/THIRDPARTY has been merged)
$ git pull origin master
$ cd ..
$ git status
# Make sure that you see "modified:  THIRDPARTY (new commits)"
$ git commit -am "updated submodule"
```

1.23 Write and Label GitHub Issues

1.23.1 Create an Issue

To create an issue:

1. Go to the [OpenMS codebase](#).
2. Submit an [issue](#).

The issue will be listed under **Issues**.

1.23.2 Label an Issue

To label an issue:

1. On the right of the screen, select the cog icon under **Labels**.
2. Choose a label from the list. Normally, an issue can have one or more of the following labels:
 - **defect**: A defect refers to a bug in OpenMS. This is a high priority issue.
 - **enhancement**: An enhancement refers to a feature idea to enhance the current OpenMS code. This is a medium priority issue.
 - **task**: A task refers to a single piece of work that a developer can undertake. This is a medium priority issue.
 - **refactoring**: A refactoring issue refers to a suggestion to streamline the code without changing how the code function.
 - **question**: A question could trigger to a discussion about tools, parameters and scientific tasks.

1.24 Adding New Tool to The TOPP suite

1.24.1 The OpenMS pipeline (TOPP)

Any tool that is written with the OpenMS library can easily be made into a TOPP tool by simply using the OpenMS command line parser which is able to parse ParamXML, a powerful XML based description of the tool. Hence most analysis algorithms in OpenMS are available as a stand-alone tool which can be called on the command line or integrated into workflow engines via the CTD mechanism. A current list of TOPP tools can be found in [the documentation](#).

1.24.2 What do I have to do to add a new TOPP tool?

The recommended way is to inherit from the class TOPPBase as in existing TOPP tools (sources available in `/src/topp/`). This will add command line parsing functionality to your tool as described in the TOPP section of this page.

- Add the code to `src/topp/` and register it in `src/topp/executables.cmake`
- Add your tool (with the correct category) to `getTOPPToolList()` in `src/openms/source/APPLICATIONS/ToolHandler.cpp`. This creates a doxygen page with the `-help` output of the tool (using TOPPDocumenter). This page must be included at the end of the doxygen documentation of your tool (see other tools for an example).
- Add it to the TOPP docu page (in `doc/doxygen/public/TOPP.doxygen`)
- Add the name to `src/topp/executables.cmake`
- Write a TOPP test (add it to `src/tests/topp/CMakeLists.txt`)

Warning: Handle any kind of input files to your TOPP tool via command line flags and use the `${DATA_DIR_TOPP}` prefix. Use ini-files to specify output-files, but not input-files. Doing otherwise will break out-of-source builds.

Hint: add `-test` to the call of your TOPP tool and also create the expected output that you put in `src/tests/topp` with that flag active. The flag ensures that UniqueId's, dates etc are equal no matter where and when the tool is run.

1.24.3 What do I have to do to add a new UTILS tool?

- Add the code to `src/utils/` and register it in `src/utils/executables.cmake`.
- Add your tool to `getUtilList()` in `src/openms/source/APPLICATIONS/ToolHandler.cpp`. This creates a doxygen page with the `-help` output of the tool (using TOPPDocumenter). This page must be included at the end of the doxygen documentation of your tool (see other tools for an example).
- Add it to the UTILS docu page (in `doc/doxygen/public/UTILS.doxygen`)
- Write a test (this is optional for UTILS). See TOPP tools above and add the test to the bottom of `src/tests/topp/CMakeLists.txt`.

1.24.4 I want to implement a new file adapter. What is to be done?

First, add a file adapter class to the `include/OpenMS/FORMAT/` and `source/FORMAT/` folders. The file adapter should implement a default constructor, a load method and a store method. Make sure your code conforms to the OpenMS Coding conventions. For automatic file type recognition, you need to

- register your new file type at the Type enum in `/include/OpenMS/FORMAT/FileTypes.h`,
- flag the file type as supported in the `isSupported` method of `/source/FORMAT/FileHandler.C`
- register the file extension in the `getTypeByFileName` method of `/source/FORMAT/FileHandler.C`

If the new file is a peak or feature file format you should also add it to `loadExperiment` or `loadFeatures`, respectively, of the FileHandler class. To add the file format to the TOPPView open dialog, you have to modify the file `/source/APPLICATIONS/TOPPViewBase.C`.

- Add the file extensions to the `filter_all` and `filter_single` variables of the `getFileList_` method.

To add your format to TOPP applications:

- add the file extension to the extensions list of the respective parameter:

```
e.g. setValidStrings_("in_type", StringList::create("mzData,mzXML,mzML")); in_
↪FileInfo
```

1.24.5 How to create an icon file for a TOPP tool under Windows?

- Create an .ico file: first, you need some graphics program (The GIMP is recommended) think of a motive and remind yourself that you have limited space. Create at least a 16x16, 32x32, 48x48 and 64x64 pixel version and save each of them in a separate layer of the respective size. Do not add any larger sized layers, since Win XP will not display any icon then. When saving the image as type .ico the GIMP will ask you for the color depth of each layer. As it is recommended to have multiple color depths of each icon-size, go back to the layers and duplicate each layer twice. That should give you 12 layers. Now, save the image as .ico (e.g. TOPPView.ico) file, giving each group of equal sized layers a 32 bit (8 bit transparency), 8 bit (1 bit transparency), 4 bit (1 bit transparency) color depth.

Attention: Make sure to assign the higher color depth to the upper layers as Windows will not pick the highest possible color otherwise.

- Create a resource file: Create a text file named .rc (e.g. TOPPView.rc) Insert the following line: 101 ICON "TOPPView.ico", replacing TOPPView with your binary name. Put both files in `OpenMS/source/APPLICATIONS/TOPP/` (similar files for other TOPP tools already present). Re-run cmake and re-link your TOPP tool.

Voila. You should have an iconized TOPP tool.

1.24.6 Develop your Tool in an external project using OpenMS

To include the OpenMS library in one of your projects, we recommend to have a look at a small emulated external project in our repository. We strongly suggest to use CMake for building your project together with OpenMS to make use of the macros and environment information generated during the build of the OpenMS library.

1.24.7 The Common Tool Description (CTD)

The CTD is a format developed from the OpenMS team to allow the user to use TOPP tools also in other workflow engines. Each tool can output a CTD description of itself (the XML scheme for the CTD can be found [here](#)), which can then be used by a node generator program to generate nodes for different workflow engines. The CTD mechanism is shared by OpenMS with other mature libraries like SeqAn and BALL. An example for a node generation program are the Generic KNIME Nodes. The most complete description on how to generate your own Generic KNIME Nodes based on a CTD (e.g. from your freshly developed command line tool), can be found on the SeqAn documentation. We are working on a tutorial specifically tailored to OpenMS.

1.25 Pull Request Checklist

Before opening a pull request, check the following:

1. **Does the code build?** Execute `make` (or your build system's equivalent, e.g., `cmake --build . --target ALL_BUILD --config Release` on Windows).
2. **Do all tests pass?** To check if all tests have passed, execute `ctest`. If a test that is unrelated to your changes fails, check the [nightly builds](#) to see if the error is also in `develop`. If the error is in `develop`, [create a github issue](#).
3. **Is the code documented?** Document all new classes, including their methods and parameters. It is also recommended to document non-public members and methods.
4. **Does the code introduce changes to the API?** If the code introduces changes to the API, make sure that the documentation is up-to-date and that the Python bindings (`pyOpenMS`) still work. For each change in the C++ API, make a change in the Python API wrapper via the `pyOpenMS/pxds/` files.
5. **Have you completed regression testing?** Make sure that you include a test in the test suite for:
 - Public methods of a class
 - TOPP tools
 - Bug fixes

Make sure to:

- **Rebase before you open a pull request.** To include all recent changes, rebase your branch on `develop` before opening a pull request. If you pushed your branch to `origin` before rebasing, git will most likely tell you after the rebase that your local branch and the remote branch have diverged. If you are sure that the remote branch does not contain any local commits in the rebased version, you can safely push using `git push -f origin <branch-name>` to enforce overwrite. If not, contact your local git expert on how to get the changes into your local branch.
- **Capture similar changes in a single commit** Each commit should represent one logical unit. Consolidate multiple commits if they belong together or split single commits if they are unrelated. For example, committing code formatting together with a one-line fix makes it very hard to figure out what the fix was and which changes were inconsequential.

- **Create a pull request for a single feature or bug** If you have multiple features or fixes in a pull request, you might get asked to split your request and open multiple pull requests instead.
- **Describe what you have changed in your pull request.** When opening the pull request, give a detailed overview of what has changed and why. Include a clear rationale for the changes and add benchmark data if available. See [this request](#) for an example.

1.26 Reporting Bugs and Issues

A list of known issues in the current OpenMS release can be found [here](#). Please check if your OpenMS version matches the current version and if the bug has already been reported.

In order to report a new bug, please create a [GitHub issue](#) or [contact us](#).

Include the following information in your bug report:

1. The command line (i.e. call) including the TOPP tool and the arguments you used, or the steps you followed in a GUI tool (e.g. TOPPView) - e.g. `FeatureFinderCentroided -in myfile.mzML -out myfile.featureXML`.
2. The output of OpenMS/TOPP (or a screenshot in case of a GUI problem).
3. Operating system (e.g. “Windows XP 32 bit”, “Win 7 64 bit”, “Fedora 8 32 bit”, “macOS 10.6 64 bit”).
4. OpenMS version (e.g. “OpenMS 1.11.1”, “Revision 63082 from the SVN repository”).
5. OpenMS architecture (“32 bit” or “64 bit”)

Please provide files that we need to reproduce the bug (e.g. TOPP INI files, data files — usually mzML) via a download link, via the mailing list or by directly contacting one of the developers.

1.27 Advanced

1.27.1 Developer Guidelines For Adding New Dependent Libraries

Our dependency library philosophy

In short, requirements for adding a new library are:

- indispensable functionality
- license compatibility
- availability for all platforms

Indispensable functionality

In general, adding a new dependency library (of which we currently have more than a handful, e.g. Xerces-C or ZLib) imposes a significant integration and maintenance effort. Thus, the new library should add **indispensable functionality**. If the added value does not compensate for the overhead, alternative solutions encompass:

- write it yourself and add to the OpenMS library (i.e. its repository) directly
- write a TOPPAdapter which calls an external executable (placing the burden on the user to supply the executable)

License compatibility

OpenMS has a BSD-3 clause license and we try hard to remove dependencies of GSL-like libraries. Therefore, a new library with e.g. LGPL-2 would be prohibitive.

C++ standard compatibility

New dependency libraries needs to be compatible and therefore compilable with the same C++ standard as OpenMS.

Availability for all platforms

OpenMS has been designed for Windows, macOS, and Linux. Therefore, the new dependency library needs to be designed for these platforms.

- on **WindowsOS** this usually means, adding the new library to the Contrib in debug and release variants. In short all recent versions of Visual Studio (VS2008 and onwards) must be supported (or support must be added). This encompasses
 - a solution file (which can be either statically present or generated by a meta-system like CMake) is available
 - The library actually compiles and is linked to the **dynamic** VS-C++ runtime lib (since this is what the OpenMS lib will link to as well - combining static and dynamic links will lead to linker errors or segfaults).
- on **macOS** it should be ensured that the library can be build on recent macOS versions (> 10.10) compiled using the mac specific *libc++*. Ideally the package should be available via **HomeBrew** or **MacPorts** so we can directly use those libraries instead of shipping them via the contrib. Additionally, the MacPorts and HomeBrew formulas for building the libraries can serve as blueprints on how to compile the library in a generic setting inside the contrib which should also be present.
- on **Linux** since we (among other distributions) feature an OpenMS Debian package which requires that all dependencies of OpenMS are available as Debian package as well, the new library must be available (or made available) as Debian package or linked statically during the OpenMS packaging build.

How to add it to the contrib build

Add a CMake file to OpenMS/contrib into the `libraries.cmake` folder on how to build the library. Preferably of course the library supports building with CMake (see Xerces) which makes the script really easy. It should support static and dynamic builds on every platform. Add the compile flag for position independent code (e.g. `-fpic`) in the static version. Add patches in the *patches* folder and call them with the macros in the `macros.cmake` file. Create patches with `diff -Naur original_file my_file > patch.txt`. If there are problems during applying a patch, make sure to double check filepaths in the head of the patch and the call of the patching macro in CMake.

- All the libraries need to go into (e.g. copied/installed/moved) to `$buildfolder/lib`
- All the headers under `$buildfolder/include/$libraryname` (the only exception to leave out the library name subfolder is when the `Find$libraryname.cmake` does not support this subfolder e.g. because system libraries are not structured like this, see boost).
- All needed files into `$buildfolder/share/$libraryname`

Then test the build on your platform including a subsequent build of OpenMS using that library. Submit a pull request to OpenMS/contrib. Submit a pull request to OpenMS/OpenMS that updates the contrib submodule. Make sure the libraries are correctly shipped in pyOpenMS and the packages (especially dynamic libraries and especially on Windows).

1.27.2 Custom Compilation of OpenMS

To compile with self built compilers and non default standard libraries, follow listed steps.

To choose any specific compiler, instead of the system default, add the whole path to these options for the cmake call:

```
cmake -DCMAKE_C_COMPILER=/path/to/c-compiler/binary/gcc -DCMAKE_CXX_COMPILER=/path/to/c++compiler/binary/g++
```

```
cmake -DCMAKE_C_COMPILER=/path/to/c-compiler/binary/clang -DCMAKE_CXX_COMPILER=/path/to/c++compiler/binary/clang
```

To compile OpenMS with clang and a specific GCC stdlib, instead of the system default one:

Use this cmake option to specify an additional compiling option for clang:

```
cmake -DMY_CXX_FLAGS="--gcc-toolchain=/path/to/gcc"
```

with the path to the top gcc directory (containing the directory lib64) to the cmake call.

Warning: This combination does not work for all versions of clang and gcc.

- Clang 9.0.0 and GCC 4.8.5 stdlib does not work!
- Clang 9.0.0 and GCC 9.2.0 stdlib does not work!
- Clang 9.0.0 and GCC 8.3.0 stdlib compiles, but some tests fail.
- Clang 6.0.0 and GCC 7.4.0 stdlib (Ubuntu 18.04 default versions) works

1.27.3 Build Custom KNIME Plugin

This page is under construction.

1.28 OpenMS Installers

Platform	Name
Windows	OpenMS-3.0.0-Win64.exe
macOS	OpenMS-3.0.0-macOS.dmg
GNU/Linux	OpenMS-3.0.0-Debian-Linux-x86_64.deb
Source	OpenMS-3.0.0-src.tar.gz

1.29 Workflows

Workflow	Description	Download Link
ProteomicsLFQ_tool_a	Label-free identification and quantification using the comet search engine, the ProteomicsLFQ tool and statistical down-stream processing using MSstats. Compared to the other proteomics LFQ workflows, it is less complex as it combines quantification and inference steps in a single ProteomicLFQ tool.	Download
DIAMetAlyzer	Metabolomics assay library construction with decoy generation from DDA data and targeted DIA analysis using OpenSWATH and pyprophet for statistical validation.	Download
Identification_quant	Identification and quantification for isobaric experiments using MSGFPlus as search engine, epifany for inference and MSstatsTMT for statistical down-stream analysis.	Download
labelfree_with_prote	Label-free with protein quantification steps implemented using individual OpenMS tools	Download
Metabolite_Adduct_Gr	Quantification and identification via accurate mass based on multiple adduct grouping steps (adducts, neutral losses).	Download
Metabolite_DeNovoID	Quantification and identification via adduct grouping and de-novo identification using SIRIUS/CSI:FingerID.	Download
Metabolite_ID	Quantification and identification via accurate mass based with downstream processing and visualisation.	Download
Metabolite_SpectralI	Identification via spectral library search for small molecules.	Download
MSstats_statPostProc	Post processing workflow for using MSstats based on “Example_OneTool_ProteomicsLFQ_MSstats.knwf”	Download
MSstatsTMT	Post processing workflow for using MSstatsTMT based on “Identification_quantification_with_inference_isobaric_epifany_MSstatsTMT”.	Download
OpenSWATH	Targeted extraction and scoring of transitions in DIA data based on an (iRT) assay library.	Download
Phosphoproteomics_ID	Identification of Phosphorilation sites.	Download

1.30 OpenMS Releases

Release	Installers
Stable release	Archive Link
Release candidates	Archive Link
Nightly release	Archive Link

1.31 Other Resources

Name	Description	Download Link
Schemas	Documented schemas of the OpenMS formats	Download
iPRG2016 data	Dataset mxMLs, Fasta database, Identification file (idXML), Big Data (idXML)	Download

1.32 Contributor FAQ

The following contains answers to typical questions from contributors about OpenMS.

1.32.1 General

The following section provides general information to new contributors.

I am new to OpenMS. What should I do first?

- Check out the development version of OpenMS (see OpenMS releases and installers).
- Build OpenMS according to the installation instructions.
- Read the [OpenMS Coding Conventions](#).
- Read the *OpenMS User Tutorial*.
- Create a GitHub account.
- Subscribe to the [open-ms-general](#) or *contact us*.

What is the difference between an OpenMS tool and util?

A tool starts its lifecycle in UTILS and may exist without being thoroughly tested. Tools may be promoted from UTILS to TOOLS if they are stable enough, are fully tested, fully documented, and a test workflow exists.

I have written a class for OpenMS. What should I do?

Follow the [OpenMS coding conventions](#).

Coding style (brackets, variable names, etc.) must conform to the conventions.

- The class and all the members must be documented thoroughly.
- Check your code with the tool `tools/checker.php`. Call `php tools/checker.php` for detailed instructions.

Please open a pull request and follow the *[pull request guidelines](#)*.

1.32.2 Troubleshooting

The following section provides information about how to troubleshoot common OpenMS issues.

OpenMS complains about boost not being found

CMake got confused. Set up a new build directory and try again. Build from source, deleting the `CMakeCache.txt` and `cmake` directory might help.

1.32.3 Build System

The following questions are related to the build system.

What is CMake?

CMake builds BuildSystems for different platforms, e.g. VisualStudio Solutions on Windows, Makefiles on Linux etc. This allows us to define in one central location (namely `CMakeLists.txt`) how OpenMS is build and have the platform specific stuff handled by CMake. View the [cmake website](#) for more information.

How do I use CMake?

See Installation instructions for your platform. In general, call `CMake(.exe)` with some parameters to create the native build-system. Afterwards, (but usually) don't have to edit the current configuration using a GUI named `ccmake` (or `CMake-GUI` in Windows), which ships with CMake).

Note: Whenever `ccmake` is mentioned in this document, substitute this by `CMake-GUI` if your OS is Windows. Edit the `CMakeCache.txt` file directly.

How do I generate a build-system for Eclipse, KDevelop, CodeBlocks etc?

Type `cmake` into a console. This will list the available code generators available on your platform, pass them to CMake using the `-G` option.

How do I add a new class to the build system?

1. Create the new class in the corresponding sub-folder of the sub-project. The header has to be created in `src/<sub-project>/include/OpenMS` and the cpp file in `src/<sub-project>/source`, e.g., `src/openms/include/OpenMS/FORMAT/NewFileFormat.h` and `src/openms/source/FORMAT/NewFileFormat.cpp`.
2. Add both to the respective `sources.cmake` file in the same directory (e.g., `src/openms/source/FORMAT/` and `src/openms/include/OpenMS/FORMAT/`).
3. Add the corresponding class test to `src/tests/class_tests/<sub-project>/` (e.g., `src/tests/class_tests/openms/source/NewFileFormat_test.cpp`).
4. Add the test to the `executables.cmake` file in the test folder (e.g., `src/tests/class_tests/openms/executables.cmake`).
5. Add them to git by using the command `git add`.

How do I add a new directory to the build system?

1. Create two new `sources.cmake` files (one for `src/<sub-project>/include/OpenMS/MYDIR`, one for `src/<sub-project>/source/MYDIR`), using existing `sources.cmake` files as template.
2. Add the new `sources.cmake` files to `src/<sub-project>/includes.cmake`
3. If you created a new directory directly under `src/openms/source`, then have a look at `src/tests/class_tests/openms/executables.cmake`.
4. Add a new section that makes the unit testing system aware of the new (upcoming) tests.
5. Look at the very bottom and augment `TEST_executables`.
6. Add a new group target to `src/tests/class_tests/openms/CMakeLists.txt`.

1.32.4 Debugging

The following section provides information about how to debug your code.

How do I run a single test?

Execute an OpenMS class test using the CTest regular expressions:

```
$ ctest -V -R "^<class>_test"

# To build a class test, call the respective make target in ./source/TEST:

$ make <class>_test
```

To run a TOPP test, use:

```
$ ctest -V -R "TOPP_<tool>"
```

To build the tool, use:

```
$ make <tool>
```

How do I debug uncaught exceptions?

Dump a core if an uncaught exception occurs, by setting the environment variable `OPENMS_DUMP_CORE`.

Each time an uncaught exception occurs, the `OPENMS_DUMP_CORE` variable is checked and a segmentation fault is caused, if it is set.

(Linux) How can I set breakpoints in gdb to debug OpenMS?

Debug the TOPPView application to stop at line 341 of SpectrumMDIWindow.C.

1. Enter the following in your terminal:

```
Run gdb:
shell> gdb TOPPView
```

2. Start the application (and close it):

```
gdb> run [arguments]
```

3. Set the breakpoint:

```
gdb> break SpectrumMDIWindow.C:341
```

4. Start the application again (with the same arguments):

```
gdb> run
```

1.32.5 Doxygen Documentation

Where can I find the definition of the main page?

Find a definition of the main page [here](#).

Where can I add a new module?

Add a new module [here](#).

How is the command line documentation for TOPP/UTILS tools created?

The program `OpenMS/doc/doxygen/parameters/TOPPDocumenter.cpp` creates the command line documentation for all classes that are included in the static `ToolHandler.cpp` tools list. It can be included in the documentation using the following doxygen command:

```
@verbatiminclude TOPP_<tool name>.cli
```

Test if everything worked by calling `make doc_param_internal`. The command line documentation is written to `OpenMS/doc/doxygen/parameters/output/`.

What are the important files for adding a new tutorial section?

View the following OpenMS tutorials:

- `OpenMS/doc/OpenMS_tutorial/refman_overwrite.tex.in` (for PDF tutorials)
- `OpenMS/doc/doxygen/public/OpenMS_Tutorial_html.doxygen` (for html tutorials)

View the following TOPP and TOPPView tutorials:

- `OpenMS/doc/TOPP_tutorial/refman_overwrite.tex.in` (for PDF tutorials)
- `OpenMS/doc/doxygen/public/TOPP_Tutorial_html.doxygen` (for html tutorials)

1.32.6 Bug Fixes

How do I contribute to a bug fix?

To contribute to a bug fix:

1. Submit the bug as a GitHub issue.
2. Create a feature branch (e.g. `feature/fix_missing_filename_issue_615`) from your (up-to-date) develop branch in your fork of OpenMS.
3. Fix the bug and add a test.
4. Create a pull request for your branch.
5. After approval and merge make sure the issue is closed.

1.33 Developer FAQ

The following contains answers to typical questions from developers about OpenMS.

1.33.1 General

The following section provides general information to new contributors.

I am new to OpenMS. What should I do first?

- Check out the development version of OpenMS (see website).
- Build OpenMS by following the installation instructions or *from source*.
- Read the [OpenMS Coding Conventions](#)
- Read the *[OpenMS User Tutorial](#)*.
- Create a GitHub account.
- Subscribe to the [open-ms-general](#) or *[contact-us](#)*.

What is the difference between an OpenMS tool and util?

A tool starts its lifecycle in UTILS and may exist without being thoroughly tested. Tools may be promoted from UTILS to TTOOLS if they are stable enough, are fully tested, fully documented, and a test workflow exists.

I have written a class for OpenMS. What should I do?

Follow the [OpenMS coding conventions](#).

Coding style (brackets, variable names, etc.) must conform to the conventions.

- The class and all the members should be properly documented.
- Check your code with the tool `tools/checker.php`. Call `php tools/checker.php` for detailed instructions.

Please open a pull request and follow the *[pull request guidelines](#)*.

Can I use QT designer to create GUI widgets?

Yes. Create a class called `Widget`: Create `.ui`-File with QT designer and store it as `Widget.ui`, add the class to `sources.cmake`. From the `.ui`-File the file `include/OpenMS/VISUAL/UIC/ClassTemplate.h` is generated by the build system.

Note: Do not check in this file, as it is generated automatically when needed.

Derive the class `Widget` from `WidgetTemplate`. For further details, see the `Widget.h` and `Widget.cpp` files.

Can the `START_SECTION`-macro not handle template methods that have two or more arguments?

Insert round brackets around the method declaration.

Where can I find the binary installers created?

View the binary installers at the [build archive](#). Please verify the creation date of the individual installers, as there may have been an error while creating the installer.

1.33.2 Troubleshooting

The following section provides information about how to troubleshoot common OpenMS issues.

OpenMS complains about boost not being found but I'm sure its there

CMake got confused. Set up a new build directory and try again. If you build from source (not recommended), deleting the `CMakeCache.txt` and `cmake` directory might help.

1.33.3 Build System

The following questions are related to the build system.

What is CMake?

CMake builds BuildSystems for different platforms, e.g. VisualStudio Solutions on Windows, Makefiles on Linux etc. This allows to define in one central location (namely `CMakeLists.txt`) how OpenMS is build and have the platform specific stuff handled by CMake.

View the [cmake website](#) for more information.

How do I use CMake?

See Installation instructions for your platform. In general, call `CMake(.exe)` with some parameters to create the native build-system.

Tip: whenever `ccmake` is mentioned in this document, substitute this by `CMake-GUI` if your OS is Windows. Edit the `CMakeCache.txt` file directly.

How do I generate a build-system for Eclipse, KDevelop, CodeBlocks etc?

Type `cmake` into a console. This will list the available code generators available on your platform; use them with `CMake` using the `-G` option.

What are user definable CMake cache variables?

They allow the user to pass options to `CMake` which will influence the build system. The most important option which should be given when calling `CMake.exe` is:

`CMAKE_FIND_ROOT_PATH`, which is where `CMake` will search for additional libraries if they are not found in the default system paths. By default we add `OpenMS/contrib`.

If you have installed all libraries on your system already, there is no need to change `CMAKE_FIND_ROOT_PATH`. For `contrib` libraries, set the variable `CMAKE_FIND_ROOT_PATH`.

On Windows, `contrib` folder is required, as there are no system developer packages. To pass this variable to `CMake` use the `-D` switch e.g. `cmake -D CMAKE_FIND_ROOT_PATH:PATH="D:\\somepath\\contrib"`.

Everything else can be edited using `ccmake` afterwards.

The following options are of interest:

- `CMAKE_BUILD_TYPE` To build Debug or Release version of OpenMS. Release is the default.
- `CMAKE_FIND_ROOT_PATH` The path to the `contrib` libraries.

Tip: Provide more then one value here (e.g., `-D CMAKE_FIND_ROOT_PATH="/path/to/contrib;/usr/"` will search in your `contrib` path and in `/usr` for the required libraries)

- `STL_DEBUG` Enables STL debug mode.
- `DB_TEST` (deprecated) Enables database testing.
- `QT_DB_PLUGIN` (deprecated) Defines the db plugin used by Qt.

View the description for each option by calling `ccmake`.

Can I use another solver other than GLPK?

Other solvers can be used, but by default, the build system only links to GLPK (this is how OpenMS binary packages must be built). To use another solver, use `cmake ... -D USE_COINOR=1 ...` and refer to the documentation of the `LPWrapper` class.

How do I switch to debug or release configuration?

For Makefile generators (typically on Linux), set the `CMAKE_BUILD_TYPE` variable to either `Debug` or `Release` by calling `ccmake`. For Visual Studio, this is not necessary as all configurations are generated and choose the one you like within the IDE itself. The ‘Debug’ configuration enabled debug information. The ‘Release’ configuration disables debug information and enables optimisation.

I changed the `contrib` path, but re-running CMake won’t change the library paths?

Once a library is found and its location is stored in a cache variable, it will only be searched again if the corresponding entry in the cache file is set to `false`.

Warning: If you delete the `CMakeCache.txt`, all other custom settings will be lost.

The most useful targets will be shown to you by calling the targets `target`, i.e. `make targets`.

CMake can’t seem to find a Qt library (usually `QtCore`). What now?

CMake finds QT by looking for `qmake` in your `PATH` or for the Environment Variable `QTDIR`. Set these accordingly.

Make sure there is no second installation of Qt (especially the MinGW version) in your local environment.

Warning: This might lead CMake to the wrong path (it’s searching for the `Qt*.lib` files). You should only move or delete the offending Qt version if you know what you are doing!

A save workaround is to edit the `CMakeCache` file (e.g. via `ccmake`) and set all paths relating to QT (e.g. `QT_LIBRARY_DIR`) manually.

(Windows) What version of Visual Studio should I use?

It is recommended to use the latest version. Get the latest CMake, as its generator needs to support your VS. If your VS is too new and there is no CMake for that yet, you’re gonna be faced with a lot of conversion issues. This happens whenever the Build-System calls CMake (which can be quite often, e.g., after changes to `CMakeLists.txt`).

How do I add a new class to the build system?

1. Create the new class in the corresponding sub-folder of the sub-project. The header has to be created in `src/<sub-project>/include/OpenMS` and the `.cpp` file in `src/<sub-project>/source`, e.g., `src/openms/include/OpenMS/FORMAT/NewFileFormat.h` and `src/openms/source/FORMAT/NewFileFormat.cpp`.
2. Add both to the respective `sources.cmake` file in the same directory (e.g., `src/openms/source/FORMAT/` and `src/openms/include/OpenMS/FORMAT/`).
3. Add the corresponding class test to `src/tests/class_tests/<sub-project>/` (e.g., `src/tests/class_tests/openms/source/NewFileFormat_test.cpp`).
4. Add the test to the `executables.cmake` file in the test folder (e.g., `src/tests/class_tests/openms/executables.cmake`).
5. Add them to git by using the command `git add`.

How do I add a new directory to the build system?

1. Create two new `sources.cmake` files (one for `src/<sub-project>/include/OpenMS/MYDIR`, one for `src/<sub-project>/source/MYDIR`), using existing `sources.cmake` files as template.
2. Add the new `sources.cmake` files to `src/<sub-project>/includes.cmake`
3. If you created a new directory directly under `src/openms/source`, then have a look at `src/tests/class_tests/openms/executables.cmake`.
4. Add a new section that makes the unit testing system aware of the new (upcoming) tests.
5. Look at the very bottom and augment `TEST_executables`.
6. Add a new group target to `src/tests/class_tests/openms/CMakeLists.txt`.

How can I speed up the compile process of OpenMS?

To speed up the compile process of OpenMS, use several threads. If you have several processors/cores, build OpenMS classes/tests and TOPP tools in several threads. On Linux, use the `make` option `-j`: `make -j8 OpenMS TOPP test_build`.

On Windows, Visual Studio solution files are automatically build with the `/MP` flag, such that Visual Studio uses all available cores of the machine.

1.33.4 Release

View [preparation of a new OpenMS release](#) to learn more about contributing to releases.

1.33.5 Working in Integrated Development Environments (IDEs)

Why are there no `source/TEST` and `source/APPLICATIONS/TOPP|UTILS` folder?

All source files added to an IDE are associated with their targets. Find the source files for each test within its own subproject. The same is true for the TOPP and UTILS classes.

I'm getting the error "Error C2471: cannot update program database"

This is a bug in Visual Studio and there is a [bug fix](#). Only apply it if you encounter the error. The bug fix might have unwanted side effects!

Visual Studio can't read the clang-format file.

Depending on the Visual Studio version it might get an error like `Error while formatting with ClangFormat`. This is because Visual Studio is using an outdated version of clang-format. Unfortunately there is no easy way to update this using Visual Studio itself. There is a plugin provided by LLVM designed to fix this problem, but the plugin doesn't work with every Visual Studio version. In that case, update clang-format manually using the pre-build clang-format binary. Both the binary and a link to the plugin can be found [here](#). To update clang-format download the binary and exchange it with the clang-format binary in your Visual Studio folder. For Visual Studio 17 and 19 it should be located at: `C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\Llvm\bin`.

The indexer gets stuck at some file which `#includes seqan`

It seems that SeqAn code is just too confusing for older eclipse C++ indexers. You should upgrade to eclipse galileo (CDT 6.0.x). Also, increase the available memory limit in `eclipse.ini`, e.g. `-Xmx1024m` for one gig.

The parser is confused after `OPENMS_DLLAPI` and does not recognize standard C++ headers

Go to `Project -> Properties -> C/C++ Include Paths and Preprocessor Symbols -> Add Preprocessor symbol -> "OPENMS_DLLAPI="`. This tells eclipse that the macro is defined empty. In the same dialog add an external include path to e.g. `/usr/include/c++/4.3.3/`, etc. The issue with C++ headers was fixed in the latest galileo release.

Hints to resolve the `OPENMS_DLLAPI` issue using the cmake generator are welcome!

1.33.6 Debugging

The following section provides information about how to debug your code.

How do I debug uncaught exceptions?

Dump a core if an uncaught exception occurs, by setting the environment variable `OPENMS_DUMP_CORE`.

Each time an uncaught exception occurs, the `OPENMS_DUMP_CORE` variable is checked and a segmentation fault is caused, if it is set.

(Linux) Why is no core dumped, although a fatal error occurred?

The `ulimit -c unlimited` command. It sets the maximum size of a core to unlimited.

Warning: We observed that, on some systems, no core is dumped even if the size of the core file is set to unlimited. We are not sure what causes this problem.

(Linux) How can I set breakpoints in gdb to debug OpenMS?

Imagine you want to debug the TOPPView application and you want it to stop at line 341 of SpectrumMDIWindow.C.

1. Enter the following in your terminal:

```
Run gdb:  
shell> gdb TOPPView
```

2. Start the application (and close it):

```
gdb> run [arguments]
```

3. Set the breakpoint:

```
gdb> break SpectrumMDIWindow.C:341
```

4. Start the application again (with the same arguments):

```
gdb> run
```

How can I find out which shared libraries are used by an application?

Linux: Use `ldd`.

Windows (Visual studio console): See [Dependency Walker](#) (use x86 for 32 bit builds and the x64 version for 64bit builds. Using the wrong version of depends.exe will give the wrong results) or `dumpbin /DEPENDENTS OpenMS.dll`.

How can I get a list of the symbols defined in a (shared) library or object file?

Linux: Use `nm <library>`.

Use `nm -C` to switch on demangling of low-level symbols into their C++ equivalent names. `nm` also accepts `.a` and `.o` files.

Windows (Visual studio console): Use `dumpbin /ALL <library>`.

Use `dumpbin` on object files (`.o`) or (shared) library files (`.lib`) or the DLL itself e.g. `dumpbin /EXPORTS OpenMS.dll`.

1.33.7 Cross-platform thoughts

OpenMS runs on three major platforms... Here are the most prominent causes of “it runs on Platform A, but not on B. What now?”

Reading or writing binary files

Reading or writing binary files causes different behaviour. Usually Linux does not make a difference between text-mode and binary-mode when reading files. This is quite different on Windows as some bytes are interpreted as EOF, which lead might to a premature end of the reading process.

If reading binary files, make sure that you explicitly state that the file is binary when opening it.

During writing in text-mode on Windows a line-break (`\n`) is expanded to (`\r\n`). Keep this in mind or use the `eol-style` property of subversion to ensure that line endings are correctly checked out on non-Windows systems.

Paths and system functions

Avoid hardcoding e.g. `String tmp_dir = "/tmp";`. This will fail on Windows. Use Qt's `QDir` to get a path to the systems temporary directory if required.

Avoid names like `uname` which are only available on Linux.

When working with files or directories, it is usually safe to use `"/"` on all platforms. Take care of spaces in directory names though. Quote paths if they are used in a system call to ensure that the subsequent interpreter takes the spaced path as a single entity.

1.33.8 Doxygen Documentation

Where can I find the definition of the main page?

Find a definition of the main page [here](#).

Where can I add a new module?

Add a new module [here](#).

How is the parameter documentation for classes derived from `DefaultParamHandler` created?

Add your class to the program `OpenMS/doc/doxygen/parameters/DefaultParamHandlerDocumenter.cpp`. This program generates a html table with the parameters. This table can then be included in the class documentation using the following doxygen command: `@htmlinclude OpenMS_<class name>.parameters`.

Note: Parameter documentation is automatically generated for TOPP/UTILS included in the static `ToolHandler.cpp` tools list.

To include TOPP/UTILS parameter documentation use following doxygen command:

```
@htmlinclude TOPP_<tool name>.parameters
```

or

```
@htmlinclude UTILS_<tool name>.parameters
```

Test if everything worked by calling `make doc_param_internal`. The parameters documentation is written to `OpenMS/doc/doxygen/parameters/output/`.

How is the command line documentation for TOPP/UTILS tools created?

The program `OpenMS/doc/doxygen/parameters/TOPPDocumenter.cpp` creates the command line documentation for all classes that are included in the static `ToolHandler.cpp` tools list. It can be included in the documentation using the following doxygen command:

```
@verbatiminclude TOPP_<tool name>.cli
```

Test if everything worked by calling `make doc_param_internal`. The command line documentation is written to `OpenMS/doc/doxygen/parameters/output/`.

1.33.9 Bug Fixes

How to contribute a bug fix?

Read *contributor quickstart guide*.

How can I profile my code?

IBM's profiler, available for all platforms (and free for academic use): Purify(Plus) and/or Quantify.

Windows: this is directly supported by Visual Studio (Depending on the edition: Team and above). Follow their documentation.

Linux:

1. Build OpenMS in debug mode (set `CMAKE_BUILD_TYPE` to Debug).
2. Call the executable with valgrind: `valgrind -tool=callgrind`.

Warning: Other processes running on the same machine can influence the profiling. Make sure your application gets enough resources (memory, CPU time).

3. Start and stop the profiling while the executable is running e.g. to skip initialization steps:
4. Start valgrind with the option `-instr-atstart=no`.
5. Call `callgrind -i [on|off]` to start/stop the profiling.
6. The output can be viewed with `kcachegrind callgrind.out`.

(Linux) How do I check my code for memory leaks?

- Build OpenMS in debug mode (set `CMAKE_BUILD_TYPE` to Debug).
- Call the executable with valgrind: `valgrind --suppressions=OpenMS/tools/valgrind/openms_external.supp -leak-check=full <executable> <parameters>`.

Common errors are:

- 'Invalid write/read ...' - Violation of container boundaries.
- '... depends on uninitialized variable' - Uninitialized variables:
- '... definitely lost' - Memory leak that has to be fixed
- '... possibly lost' - Possible memory leak, so have a look at the code

For more information see the [valgrind documentation](#).

1.34 Contact Us

Join us on [Discord](#)!

You can also contact us:

1. On the user and contributor real time [Gitter chat](#).
2. Drop us an email at user support [open-ms-general](#) mailing list.
3. To stay updated of new versions of OpenMS and releases, subscribe to [openms-announcements](#) mailing list.
4. To report a new bug, create an issue on GitHub [OpenMS](#) repository.

Say hi on [OpenMS Twitter](#)!

1.35 OpenMS Glossary

A glossary of common terms used throughout OpenMS documentation.

aerosol

An aerosol is a suspension of fine solid particles or liquid droplets in air or another gas.

atom

An atom is the smallest unit of ordinary matter that forms a chemical element.

chromatogram

A two-dimensional plot that describes the amount of analyte eluted from a chromatography versus the analyte's retention time. OpenMS represents a chromatogram using the class [MSChromatogram](#)

collision-induced dissociation (CID)

A mass spectrometry technique to induce fragmentation of selected ions in the gas phase. Also known as Collision induced dissociation.

consensus feature

Features from replicate experiments with similar retention times and m/z values are linked and considered a consensus feature. A consensus feature contains information on the common retention time and m/z values as well as intensities for each sample. OpenMS represents a consensus feature using the class [ConsensusFeature](#).

consensus map

A consensus map is a collection of *consensus features* identified from mass spectra across replicate experiments. One consensus map can contain many consensus features. OpenMS represents a consensus map using the class [ConsensusMap](#).

de novo peptide sequencing

A peptide's amino acid sequence is inferred directly from the precursor peptide mass and tandem mass spectrum (*MS/MS* or *MS³*) fragment ions, without comparison to a reference proteome.

electrospray ionization

A technique used in mass spectrometry to produce ions using an electrospray in which a high voltage is applied to a liquid to create an *aerosol*.

electrospray ionization (ESI)

A technique used in mass spectrometry to produce ions.

FASTA format

A text-based format for representing nucleotide or amino acid sequences.

feature

An LC-MS feature represents the combined isotopic mass traces of a detected chemical compound. The chromatographic peak shape of a feature is defined by the interaction of the analyte with the LC column. Each feature contains information on retention time, mass-to-charge ratio, intensity and overall quality. OpenMS represents a feature using the class [Feature](#).

feature map

A feature map is a collection of features identified in a mass spectrum from a single experiment. One feature map can contain many features. OpenMS represents a feature map using the class [FeatureMap](#).

HPLC-MS

Data produced by High performance liquid chromatography (HPLC) separates components of a mixture, whereas mass spectrometry (MS) offers the detection tools to identify them.

ion

Any *atom* or group of atoms that bears one or more positive or negative electrical charges. Positively charged are cations, negatively charged anions.

iTRAQ

Stands for Isobaric tags for relative and absolute quantitation.

KNIME

An advanced workflow editor which OpenMS provides a plugin for.

LC-MS

Liquid Chromatography-Mass Spectrometry.

Liquid chromatography

An analytical technique used to separate molecules.

LuciphorAdapter

Adapter for the LuciPHOr2: a site localisation tool of generic post-translational modifications from tandem mass spectrometry data. More information is available in the [OpenMS API reference documentation](#).

m/z

mass to charge ratio.

Mascot

Identifies peptides in MS/MS spectra via Mascot. Please find more information in the [TOPP Documentation](#).

Mass

Mass is a measure of the amount of matter that an object contains. In comparison to often used term weight, which is a measure of the force of gravity on that object.

mass spectrometry

An analytical technique used to identify and quantify molecules of interest.

mass spectrum

A mass spectrum is a plot of the ion signal as a function of the mass-to-charge ratio. A mass spectrum is produced by a single mass spectrometry run. These spectra are used to determine the elemental or isotopic signature of a sample, the masses of particles and of molecules, and to elucidate the chemical identity or structure of molecules and other chemical compounds. OpenMS represents a one dimensional mass spectrum using the class [MSSpectrum](#).

MS

Mass Spectrometry

MS(1)

First stage to get a spectra. A sample is injected into the mass spectrometer, ionized, accelerated and analyzed by mass spectrometry.

MS(2)

Ions from MS1 spectra are then selectively fragmented and analyzed by a second stage of mass spectrometry (MS2) to generate the spectra for the ion fragments.

MS/MS

Tandem mass spectrometry, MS², a technique where two or more mass analyzers are coupled together using an additional reaction step to increase their abilities to analyse chemical samples.

MS²

See [MS/MS](#).

MS³

Multi-stage Mass Spectrometry

MSEExperiment

An OpenMS class used to represent a single mass spectrometry run. [Read the documentation for further information.](#)

MSGFPlusAdapter

Adapter for the MS-GF+ protein identification (database search) engine. More information is available in the [OpenMS API reference documentation](#).

mzData

mzData was the first attempt by the Proteomics Standards Initiative (PSI) from the Human Proteome Organization (HUPO) to create a standardized format for Mass Spectrometry data.[7] This format is now deprecated, and replaced by mzML.

mzML

The mzML format is an open, XML-based format for mass spectrometer output files, developed with the full participation of vendors and researchers in order to create a single open format that would be supported by all software.

mzXML

mzXML is an open data format for storage and exchange of mass spectroscopy data, developed at the SPC/Institute for Systems Biology.

Nightly Snapshot

Untested installers and containers are known as the nightly snapshot.

octadecyl (C18)

An alkyl radical C(18)H(37) derived from an octadecane by removal of one hydrogen atom.

OpenMS API

An interface that allows developers to use OpenMS core library classes and methods.

orbitrap analyzers

In mass spectrometry, an ion trap mass analyzer consisting of an outer barrel-like electrode and a coaxial inner spindle-like electrode that traps ions in an orbital motion around the spindle. A high resolution mass spectrometry analyzer.

peak

A single raw data point in a chromatogram or a mass spectrum. OpenMS represents a peak in a chromatogram using the class [ChromatogramPeak](#). OpenMS represents a single, one-dimensional peak in a mass spectrum using the class [PeakID](#)

PepNovo

PepNovo is a de novo sequencing algorithm for [MS/MS spectra](#).

peptides

A short chain of amino acids.

proteins

Proteins are vital parts of living organisms, with many functions, for example composing the structural fibers of muscle to the enzymes that catalyze the digestion of food to synthesizing and replicating DNA.

proteomics

Proteomics is the large-scale study of proteins.

ProteoWizard

ProteoWizard is a set of open-source, cross-platform tools and libraries for proteomics data analyses. It provides a framework for unified mass spectrometry data file access and performs standard chemistry and LCMS dataset computations.

pyOpenMS

pyOpenMS is an open-source Python library for mass spectrometry, specifically for the analysis of proteomics and metabolomics data in Python. For pyOpenMS documentaion visit [this link](#).

quadrupole mass filters

A mass filter allowing one mass channel at a time to reach the detector as the mass range is scanned.

retention time

retention time (RT) in liquid chromatography, is the time it takes for a separated analyte to move through the stationary phase.

RT

Retention time.

SILAC

Stands for Stable isotope labeling using amino acids in cell culture.

spectra

Plural of spectrum.

SRM

Selected reation monitoring is a mass spectrometry technique for small molecule analysis.

SWATH

Stands for Sequential acquisition of all theoretical fragment ion spectra.

time-of-flight (TOF)

A measurement of the time taken by an object, particle of wave (be it acoustic, electromagnetic, e.t.c) to travel a distance through a medium.

TMT

Tandem Mass Tag (TMT) is a mass spectrometry based system designed to identify and quantify proteins in different samples.

TOPP

The OpenMS Pipeline.

TOPP tool

OpenMS offers a vast array of TOPP tools for processing, analyzing and visualizing mass spectrometry data.

TOPP Tools

OpenMS provides a number of functions that process mass spectrometry data called TOPP tools. More information on TOPP tools can be found in the [OpenMS API reference documentation](#).

TOPPAS

An assistant for GUI-driven TOPP workflow design. It is recommended to use OpenMS through the KNIME plugins.

TOPPView

TOPPView is a viewer for MS and HPLC-MS data. More information is available in [TOPPView documentation](#).

UTILS

Besides *TOPP*, OpenMS offers a range of other tools. They are not included in *TOPP* as they are not part of typical analysis pipelines. More information is present in [OpenMS UTILS Documentation](#).

INDICES AND TABLES

- `genindex`
- `search`

A

aerosol, [279](#)
atom, [279](#)

C

chromatogram, [279](#)
collision-induced dissociation (*CID*), [279](#)
consensus feature, [279](#)
consensus map, [279](#)

D

de novo peptide sequencing, [279](#)

E

electrospray ionization, [279](#)
electrospray ionization (*ESI*), [279](#)

F

FASTA format, [279](#)
feature, [280](#)
feature map, [280](#)

H

HPLC-MS, [280](#)

I

ion, [280](#)
iTRAQ, [280](#)

K

KNIME, [280](#)

L

LC-MS, [280](#)
Liquid chromatography, [280](#)
LuciphorAdapter, [280](#)

M

m/z, [280](#)
Mascot, [280](#)
Mass, [280](#)

mass spectrometry, [280](#)
mass spectrum, [280](#)
MS, [280](#)
MS(1), [280](#)
MS(2), [281](#)
MS/MS, [281](#)
MS², [281](#)
MS³, [281](#)
MSEExperiment, [281](#)
MSGFPlusAdapter, [281](#)
mzData, [281](#)
mzML, [281](#)
mzXML, [281](#)

N

Nightly Snapshot, [281](#)

O

octadecyl (*C18*), [281](#)
OpenMS API, [281](#)
orbitrap analyzers, [281](#)

P

peak, [281](#)
PepNovo, [281](#)
peptides, [281](#)
proteins, [282](#)
proteomics, [282](#)
ProteoWizard, [282](#)
pyOpenMS, [282](#)

Q

quadrupole mass filters, [282](#)

R

retention time, [282](#)
RT, [282](#)

S

SILAC, [282](#)
spectra, [282](#)

SRM, [282](#)
SWATH, [282](#)

T

time-of-flight (*TOF*), [282](#)
TMT, [282](#)
TOPP, [282](#)
TOPP tool, [282](#)
TOPP Tools, [282](#)
TOPPAS, [282](#)
TOPPView, [282](#)

U

UTILS, [283](#)