OpenMS Tutorial

Version: 2.4.0-GIT-NOTFOUND-GIT-NOTFOUND

Contents

1 Introduction

1.1 General Information

Mass spectrometry (MS) is an essential analytical technique for high-throughput analysis in proteomics and metabolomics. The development of new separation techniques, precise mass analyzers and experimental protocols is a very active field of research. This leads to more complex experimental setups yielding ever increasing amounts of data. Consequently, analysis of the data is currently often the bottleneck for experimental studies. Although software tools for many data analysis tasks are available today, they are often hard to combine with each other or not flexible enough to allow for rapid prototyping of a new analysis workflow.

OpenMS, a software framework for rapid **application and method development** in mass spectrometry has been designed to be portable, easy-to-use, and robust while offering a rich functionality ranging from basic data structures to sophisticated algorithms for data analysis (https://www.nature.com/nmeth/journal/v13/n9/abs/nmeth.↩
3959.html).

Ease of use: OpenMS follows the **object-oriented** programming paradigm, which aims at mapping real-world entities to comprehensible data structures and interfaces. OpenMS enforces a **coding style** that ensures consistent names of classes, methods and member variables which increases the usability as a software library. Another important feature of a software framework is documentation. We decided to use **doxygen** (www.doxygen.org/) to generate the class documentation from the **source code**, which ensures consistency of code and documentation. The documentation is generated in HTML format making it easy to read with a web browser.

Robustness: Robustness of algorithms is essential if a new method will be applied routinely to large scale datasets. Typically, there is a trade-off between performance and robustness. OpenMS tries to address both issues equally. In general, we try to tolerate recoverable errors, e.g. files that do not entirely fulfill the format specifications. On the other hand, **exceptions** are used to handle fatal errors. To check for correctness, more than 1000 **unit tests** are implemented in total, covering public methods of classes. These tests check the behavior for both valid and invalid use. Additionally, **preprocessor macros** are used to enable additional consistency checks in debug mode, enforce **pre- and post-conditions**, and are then disabled in productive mode for performance reasons.

Extensibility: Since OpenMS is based on several **external libraries** it is designed for the integration of external code. All classes are encapsulated in the OpenMS namespace to avoid symbol clashes with other libraries. Through the use of **C++ templates**, many data structures are adaptable to specific use cases. Also, OpenMS supports **standard formats** and is itself open-source software. The use of standard formats ensures that applications developed with OpenMS can be easily integrated into existing analysis pipelines. OpenMS source code is released under the permissive **BSD 3 license** and located on **GitHub**, a repository for open-source software. This allows users to participate in the project and to contribute to the code base.

Scriptable: OpenMS allows exposing its functionality through python bindings (**pyOpenMS**). This eases the rapid development of algorithms in Python that later can be translated to C++. Please see our pyOpenMS documentation for a description and walk-through of the pyOpenMS capabilities.

Portability: OpenMS supports **Windows**, **Linux**, and **OS X** platforms.

1.2 The structure of the OpenMS Framework

The following image shows the overall structure of OpenMS:

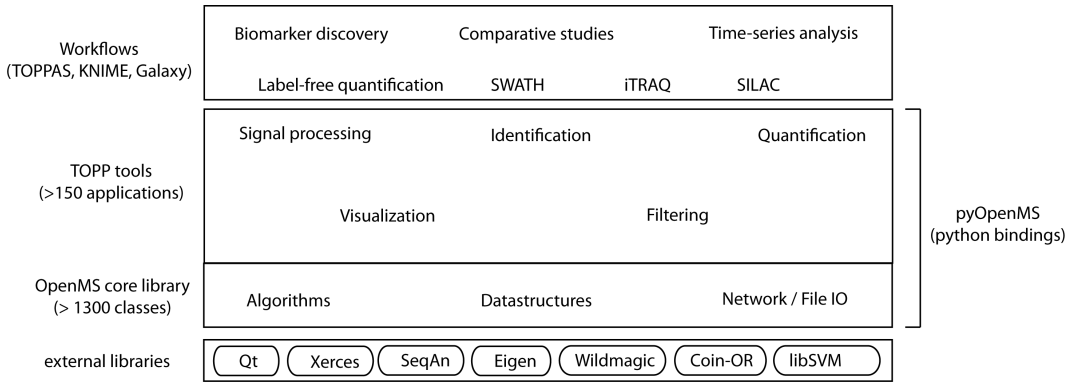

Figure 1: Overall design of OpenMS.

The structure of the OpenMS framework.

The OpenMS software framework consists of three main layers:

- **OpenMS Library:** the object-oriented OpenMS core library contains over 1,300 classes and is built on modern C++ infrastructure with native compiler support on Windows, Linux and OS X. The classes are representing core concepts in mass spectrometry as well as the corresponding ontologies defined by the Human Proteome Organization Proteomics Standard Initiative (HUPO-PSI).

- **Scripting:** a well-defined Python API offers scripting for rapid software prototyping and interactive data exploration by researchers with advanced scripting skills. The pyOpenMS interactive Python interface, providing easy integration of the OpenMS library with other scientific Python libraries.

- **TOPP tools:** a set of pre-built tools covering most core tasks in computational mass spectrometry. These tools are created using the OpenMS library. These tools form the building blocks that can be chained together to form complex workflows.

- **Workflow:** a set of over 185 different tools for common mass spectrometric tasks can be accessed by routine users through the KNIME, and Galaxy workflow systems.

Each level of increasing abstraction provides better usability, but limits the extensibility as the Python and workflow levels only have access to the exposed Python API or the available set of TOPP tools respectively. Increasing abstraction, however, makes it easier to design and execute complex analyses, even across multiple omics types. By following a layered design the different needs of bioinformaticians and life scientists are addressed.

1.3 Developing with OpenMS

Before we get started developing with OpenMS, we would like to point to some information on the development model and conventions we follow to maintain a coherent code base.

Development model

OpenMS follows the Gitflow development workflow which is excellently describedI `here`. Additionally we encourage every developer (even if he is eligible to push directly to OpenMS) to create his own fork (e.g. username). The GitHub people provide superb documentation on `forking` and how to keep your fork `up-to-date`. With your own fork you can follow the Gitflow development model directly, but instead of merging into "develop" in

your own fork you can open a pull request. Before opening the pull request, please check the checklist. Some more details and tips are collected here.

Conventions
See the manual for proper coding style: Coding conventions also see: C++ Guide. We automatically test for common coding convention violations using a modified version of cpplint. Style testing can be enabled using CMake options. We also provide a configuration file for Uncrustify for automated style corrections (see "tools/uncrustify.cfg").

Commit Messages
In order to ease the creation of a CHANGELOG we use a defined format for our commit messages. See the manual for proper commit messages: How to write commit messages.

Automated Unit Tests
Pull requests are automatically tested using our continuous integration platform. In addition we perform nightly test runs covering different platforms. Even if everything compiled well on your machine and all tests passed, please check if you broke another platform on the next day. Nightly tests: CDASH

Experimental Installers
We automatically build installers for different platforms. These usually contain unstable or partially untested code - so use them at your own risk. The nightly (unstable) installers are available here.

Technical Documentation
Documentation of classes and tools is automatically generated using doxygen: See the documentation for HEAD See the documentation for the latest release branch

Building OpenMS
Before you get started coding with OpenMS you need to build it for your operating system. Please follow the **build instructions** from the documentation.
Building OpenMS on GNU/Linux
Building OpenMS on Mac OS X
Building OpenMS on Windows
Note that for development purposes, you might want to set the variable *CMAKE_BUILD_TYPE* to *Debug*. Otherwise, the default *Release* will be applied and disables pre-condition and post-condition checks, and assertions.

Choice of an IDE
You are, of course, free to choose your favorite (or even no) IDE for OpenMS development but given the size of OpenMS, not all IDEs perform equally well. We have good experiences with Qt Creator on Linux and Mac, because it can directly import CMake Projects and is rather fast in indexing all files. On Windows, Visual Studio is currently the preferred solution. Additionally, you may want to try JetBrains CLion (it is free for students, teachers and open source projects). Another option is Eclipse with C++ support, which can also import CMake projects directly with the respective CMake generator.

1.4 Mass spectrometry terms

The following terms for MS-related data are used in this tutorial and the OpenMS class documentation:

- **Raw or profile peak**: a typically Gaussian shaped mass peak measured by the instrument.

- **Centroid or picked peak**: a single m/z, intensity pair as obtained after using a peak picking (also: peak centroiding) algorithm.

- **Spectrum / Scan**: a mass spectrum containing profile or centroided peaks (profile spectrum) or centroided peaks (peak spectrum). E.g. a low resolution profile (blue) and a centroided peak spectrum (pink) are shown in the figure below.

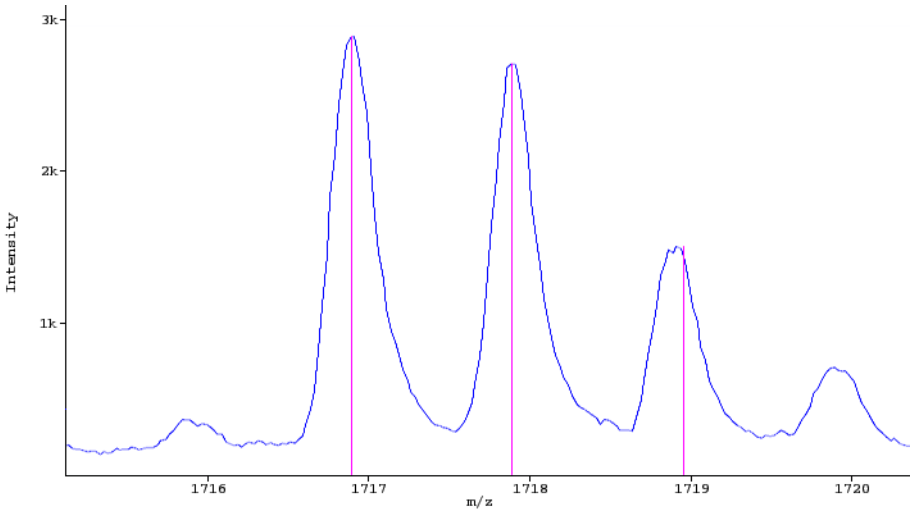

Figure 2: Part of a raw spectrum (blue) with three peaks (red)

- **(Peak or Raw) Map**: a collection of spectra of a single LC-MS run. If spectra are recorded in profile mode, we usually use the term raw map. If spectra are already centroided we usually refer to them as peak map.

- **Feature**: a signal from a chemical entity detected in an HPLC-MS experiment, typically a peptide.

The image below shows a peak map and the red circle highlights a feature.

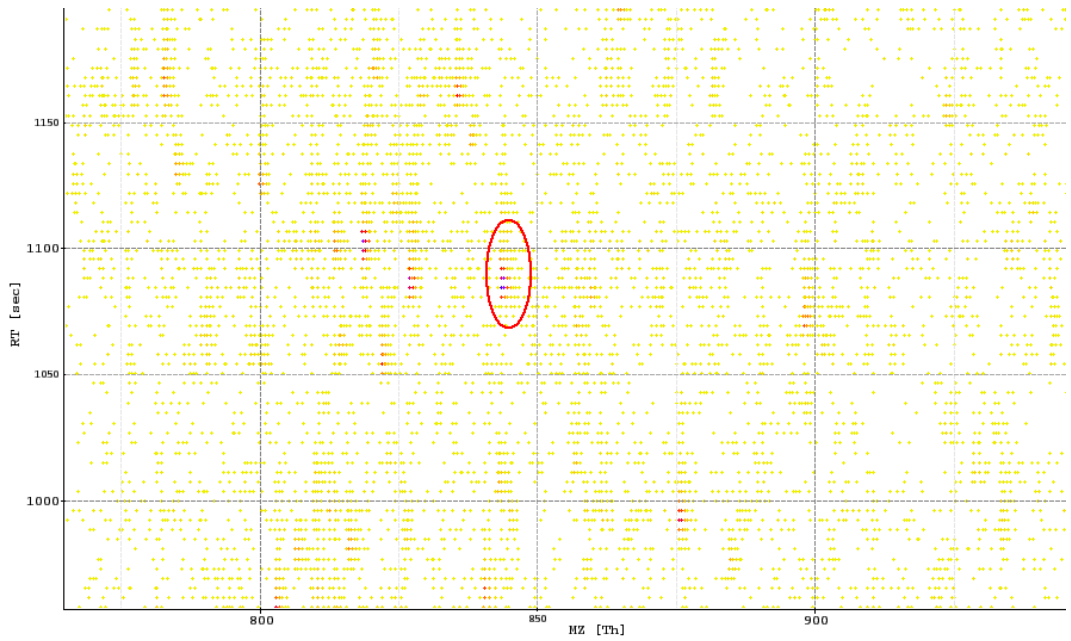

Figure 3: Peak map with a marked feature (red)

2 OpenMS Library

The extensible OpenMS library implements common mass spectrometric data processing tasks through a well defined API in C++ and Python using standardized open data formats.

2.1 Overview on Central Algorithms and Methods

OpenMS provides algorithms in many fields of computational metabolomics and proteomics.
The following list is intended to algorithm and tool developers a starting point to tools and classes relevant to their scientific question at hand. It does not include third-party tools but only tools that were implemented in OpenMS.

- Proteomics:
  - Signal processing:
    - Conversion from profile to centroided spectra (Tool PeakPickerHiRes)
    - Precursor mass correction (Tool HiResPrecursorMassCorrector)
  - Filtering:
    - Large number of basic filters applicable to different types of data (e.g., remove identified spectra, filter MS2, extract m/z ranges, . . . in Tool FileFilter and IDFilter)
  - Identification:
    - Database search:
      - Peptides (Tool SimpleSearchEngine and its classes - started simple but is, by now, rather complete peptide identification engine)
      - Protein-RNA cross-links (Tool RNPxlSearch and its classes)
      - Protein-Protein cross-links (Tool OpenPepXL)

    * Spectral library search:
        · Tool SpecLibSearcher and its classes
    * DeNovo:
        · Tool CompNovoCID and its classes
  - Quantification:
    * Peptide Feature Detection:
        · Untargeted, label-free (Tools FeatureFinderCentroided, FeatureFinderMultiplex, and its classes)
        · ID-based label-free (Tool FeatureFinderIdentification "new")
        · SILAC-labeling (Tool FeatureFinderMultiplex)
        · iTRAQ/TMT (Tool IsobaricAnalyzer)
        · Dynamically labeled (SIP) peptides (Tool MetaProSIP)
    * Retention Time Alignment:
        · Linear map alignment (Tool MapAlignerPoseClustering)
        · (Non-)linear map alignment (Tool MapAlignerIdentification "new")
    * Peptide Feature linking (matching of features between runs):
        · fast, KD-tree based linking (Tool FeatureLinkerUnlabeledKD)
        · QT based clustering and linking (Tool FeatureLinkerUnlabeledQT)
    * Protein inference:
        · WIP (currently via third-party tool FIDO and Wrapper FidoAdapter)
    * Protein Quantification:
        · Tool ProteinQuantifier
    * Targeted data extraction:
        · Analysis of data-independent acquisition or SWATH-MS data (Tool OpenSWATH)
    * Misc:
        · Theoretical spectra generators

- Metabolomics:
  - Quantification:
    * Small molecule feature detection:
        · Untargeted, label-free (Tool FeatureFinderMetabo)
    * Retention Time Alignment:
        · Linear map alignment (Tool MagAlignerPoseClustering)
    * Small molecule feature linking:
        · QT based clustering and linking (Tool FeatureLinkerUnlabeledQT)
        · fast, KD-tree based linking (Tool FeatureLinkerUnlabeledKD)
    * Adduct decharing:
        · Linear programming based determination of small molecule ion adducts and charges (Tool MetboliteAdductDecharger)
    * Targeted data extraction:
        · Analysis of data-independent acquisition or SWATH-MS data (Tool OpenSWATH)
  - Identification:
    * Spectral library search:
        · Tool MetaboliteSpectralMatcher
    * Accurate mass search:

· Tool AccurateMassSearch

- General:
  - Mass decomposition algorithms
  - Isotope pattern generators
  - Quality control (Tools QCCalculator, QCExtractor) metrics and file format (QcML)

Table 1: Directory structure of src folder (/src)

Folder	Description
openms	Source code of core library
openms_gui	Source code of GUI applications (e.g.: TOPPView)
topp	Source code of (stable) OpenMS Tools
util	Source code of (experimental) OpenMS Tools
pyOpenMS	Source files providing the python bindings
tests	Source code of class and tool tests

Table 2: Directory structure of core library (/src/openms/include/Open↩ MS)

Folder	Description
ANALYSIS	Source code of high-level analysis like PeakPicking, Quantitation, Identification, MapAlignment
APPLICATIONS	Source code for tool base and handling
CHEMISTRY	Source code dealing with Elements, Enzymes, Residues, Modifications, Isotope distributions and amino acid sequences
COMPARISON	Different scoring functions for clustering and spectra comparison
CONCEPT	OpenMS concepts (types, macros, ...)
DATASTRUCTURES	Auxiliary data structures
FILTERING	Filter
FORMAT	Source code for I/O classes and file formats
INTERFACES	Interfaces (WIP)
KERNEL	Core data structures
MATH	Source code for math functions and classes
METADATA	Source code for classes that capture metadata about a MS or HPLC-MS experiment
SIMULATION	Source code of MS simulator
SYSTEM	Source code for basic functionality (file system, stopwatch)

Folder	Description
TRANSFORMATIONS	Feature detection (MS1 label-free and isotopic labelling) and PeakPickers (centroiding algorithms)

Within the ANALYSIS folder, you can find several important tools

Table 3: Directory structure of the algorithmic part of the library (/src/openms/include/OpenMS/ANALYSIS)

Folder	Description
DECHARGING	Algorithms for de-charging (charge analysis) for peptides and metabolites
DENOVO	Algorithms for "de-novo" identification tools including CompNovo
ID	Source code dealing with identifications including ID conflict resolvers, metabolite spectrum matching and target-decoy models
MAPMATCHING	Algorithms for retention time correction and feature matching (matching between runs)
MRM	Algorithms for MRM Fragment selection
OPENSWATH	OpenSWATH algorithms for targeted, chromatogram-based analysis of MRM, SRM, PRM, DIA and SWATH-MS data
PIP	Peak intensity predictor
QUANTITATION	Algorithms for quantitative analysis including isobaric labelling
RNPXL	Algorithms for RNA cross-linking
SVM	Algorithms for SVM
TARGETED	Algorithms for targeted proteomics (MRM, SRM)
XLMS	Algorithms for Cross-link mass spectrometry

For the sake of completeness you will find a short list of the THIRDPARTY tools, which are integrated via wrappers into the OpenMS framework (usually called -Adapter e.g. SiriusAdapter)
Wrapper to third-party tools:

- Search Engines (MSGFPLUS, XTandem, OMSSA, Comet, MyriMatch)
- Protein Inference (Fido)
- Spectral Library Search (SpectraST)
- Metabolite Identification (Sirius)
- Score calibration and FDR calculation (Percolator)

2.2 Kernel Classes

The OpenMS kernel contains the data structures that store the actual MS data.
For storing the basic MS data (spectra, chromatograms, and full runs) OpenMS uses

- Peaks (Peak1D and ChromatogramPeak) stored in

- MSSpectrum and MSChromatogram, which in turn can both be stored in an
- MSExperiment

For storing quantified peptides or analytes in single MS runs, OpenMS uses so called feature maps.
The main data structures for quantitative information are

- Features (for quantitative information in MS1 maps)
- MRMFeatures (for quantitative information in XIC traces on MS1 and MS2 level)
  - which are both stored in a FeatureMap

To store quantified peptides or analytes over several MS runs, OpenMS uses so called consensus maps.

- ConsensusFeatures are stored in a
- ConsensusMap

To store identified peptides OpenMS has classes

- PeptideHit, which corresponds to a Peptide-Spectrum-Matching stored in a
- PeptideIdentification object (which is associated with a single spectrum)

Table 4: Directory structure of core library (/src/openms)

Stored Entity	Class Name
Mass Peak (m/z + intensity)	Peak1D
Elution Peak (rt + intensity)	ChromatogramPeak
Spectrum of Mass Peaks	MSSpectrum
Chromatogram of Elution Peaks	MSChromatogram
Mass trace for small molecule detection	MassTrace
Full MS run, containing both spectra and chromatograms	MSExperiment (alias PeakMap)
Feature (isotopic pattern of eluting analyte)	Feature
All features detected in an MS Run	FeatureMap
Linked / Grouped feature (e.g., same Peptide quantified in several MS runs)	ConsensusFeature
All grouped ConsensusFeatures of a multi-run experiment	ConsensusMap
Peptide Spectrum Match	PeptideHit
Identified Spectrum with one or several PSMs	PeptideIdentification
Identified Protein	ProteinHit

2.3 Peaks

OpenMS provides one-, two- and d-dimensional data points, either with or without metadata attached to them.

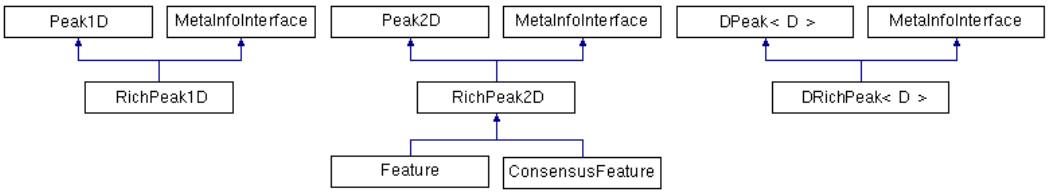

Figure 4: Data structure for MS data points

One-dimensional data points: One-dimensional data points (Peak1D) are the most important ones and used throughout OpenMS. The two-dimensional and d-dimensional data points are needed rarely and used for special purposes only. Peak1D provides getter and setter methods to store the mass-to-charge ratio and intensity.
Two-dimensional data points: The two-dimensional data points store mass-to-charge, retention time and intensity. The most prominent example we will later take a closer look at is the Feature class, which stores a two-dimensional position (m/z and RT) and intensity of the eluting peptide or analyte.
The base class of the two-dimensional data points is Peak2D. It provides the same interface as Peak1D and additional getter and setter methods for the retention time. RichPeak2D is derived from Peak2D and adds an interface for metadata. The Feature is derived from RichPeak2D and adds information about the convex hull of the feature, quality and so on.
For information on d-dimensional data points see the appendix.

2.4 Spectra

The most important container for raw/profile data and centroided peaks is MSSpectrum. The elements of a M↩ SSpectrum are peaks (Peak1D). In fact it is so common that it has its own typedef PeakSpectrum. MSSpectrum is derived from SpectrumSettings, a container for the metadata of a spectrum (e.g. precursor information). Here, only MS data handling is explained, SpectrumSettings is described in subsection meta data of a spectrum. In the following example (Tutorial_MSSpectrum.cpp) program, a MSSpectrum is filled with peaks, sorted according to mass-to-charge ratio and a selection of peak positions is displaOne-dimensional data points:

Example: Tutorial_MSSpectrum.cpp
In this example, we create MS1 spectrum at 1 minute and insert peaks with descending mass-to-charge ratios (for educational reasons). We sort the peaks according to ascending mass-to-charge ratio. Finally we print the peak positions of those peaks between 800 and 1000 Thomson. For printing all the peaks in the spectrum, we simply would have used the STL-conform methods begin() and end(). In addition to the iterator access, we can also directly access the peaks via vector indices (e.g. spectrum[0] is the first Peak1D object of the MSSpectrum).

```
#include <OpenMS/KERNEL/MSSpectrum.h>
using namespace OpenMS;
using namespace std;
int main()
{
  // Create spectrum
  MSSpectrum spectrum;
  Peak1D peak;
  for (float mz = 1500.0; mz >= 500; mz -= 100.0)
  {
    peak.setMZ(mz);
    spectrum.push_back(peak);
  }
  // Sort the peaks according to ascending mass-to-charge ratio
  spectrum.sortByPosition();
  // Iterate over spectrum of those peaks between 800 and 1000 Thomson
  for (auto it = spectrum.MZBegin(800.0); it != spectrum.MZEnd(1000.0); ++it)
  {
    cout « it->getMZ() « endl;
  }

  // Access a peak by index
  cout « spectrum[1].getMZ() « " " « spectrum[1].getIntensity() « endl;
  // ... and many more
```

```
  return 0;
}
```

2.5 Chromatograms

The most important container for targeted analysis / XIC data is MSChromatogram. The elements of a MSChromatogram are chromatogram peaks (Peak1D). MSChromatogram is derived from ChromatogramSettings, a container for the metadata of a chromatogram (e.g. containing precursor and product information), similarly to SpectrumSettings. In the following example (Tutorial_MSChromatogram.cpp) program, a MSChromatogram is filled with chromatographic peaks, sorted according to retention time and a selection of peak positions is displayed.

Example: Tutorial_MSChromatogram
Fill MSChromatogram with chromatographic peaks, sorted according to retention time

```
#include <OpenMS/KERNEL/MSChromatogram.h>
#include <OpenMS/METADATA/ChromatogramSettings.h>
#include <OpenMS/KERNEL/ChromatogramPeak.h>
using namespace OpenMS;
using namespace std;
int main()
{
  // create a chromatogram
  MSChromatogram chromatogram;
  // fill it with metadata information
  chromatogram.setNativeID("transition_300.9_188.0");
  chromatogram.getProduct().setMZ(188.0);
  chromatogram.getPrecursor().setMZ(300.9);

  // fill chromatogram with peaks
  ChromatogramPeak peak;
  peak.setIntensity(1.0);
  for (float rt = 200.0; rt >= 100; rt -= 1.0)
  {
    peak.setRT(rt);
    chromatogram.push_back(peak);
  }
  return 0;
} // end of main
```

Since much of the functionality is shared between MSChromatogram and MSSpectrum, further examples can be gathered from the MSSpectrum subsection.

2.6 Precursor

The precursor data stored along with MS/MS spectra contains invaluable information for MS/MS analysis (e.g, m/z, charge, activation mode, collision energy). This information is stored in Precursor objects that can be retrieved from each spectrum. For a complete list of functions please see the Precursor class documentation.

Example: Tutorial_Precursor
Retrieve precursor information

```
#include <OpenMS/KERNEL/MSExperiment.h>
#include <OpenMS/METADATA/Precursor.h>
#include <OpenMS/FORMAT/MzMLFile.h>
#include <OpenMS/CONCEPT/Exception.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main(int argc, const char** argv)
{

  if (argc < 2) return 1;

  // the path to the data should be given on the command line
  String tutorial_data_path(argv[1]);

  MSExperiment spectra;
  MzMLFile f;
  // load mzML from code examples folder
  f.load(tutorial_data_path + "/data/Tutorial_GaussFilter.mzML", spectra);
  // iterate over map and output MS2 precursor information
```

```
  for (auto s_it = spectra.begin(); s_it != spectra.end(); ++s_it)
  {
    // we are only interested in MS2 spectra so we skip all other levels
    if (s_it->getMSLevel() != 2) continue;
    // get a reference to the precursor information
    const MSSpectrum& spectrum = *s_it;
    const vector<Precursor>& precursors = spectrum.getPrecursors();
    // size check & throw exception if needed
    if (precursors.empty()) throw Exception::InvalidSize(__FILE__, __LINE__, OPENMS_PRETTY_FUNCTION, precursors.size());
    // get m/z and intensity of precursor
    double precursor_mz = precursors[0].getMZ();
    float precursor_int = precursors[0].getIntensity();

    // retrieve the precursor spectrum (the most recent MS1 spectrum)
    PeakMap::ConstIterator precursor_spectrum = spectra.getPrecursorSpectrum(s_it);
    double precursor_rt = precursor_spectrum->getRT();

    // output precursor information
    std::cout « " precusor m/z: " « precursor_mz
              « " intensity: " « precursor_int
              « " retention time (sec.): " « precursor_rt
              « std::endl;
   }

  return 0;
} // end of main
```

2.7 MRMTransitionGroup

The targeted analysis of SRM or DIA (SWATH-MS) type of data requires a set of targeted assays as well as raw data chromatograms. The MRMTransitionGroup class allows users to map these two types of information and store them together with identified features conveniently in a single object.

Example: Tutorial_MRMTransitionGroup

Create an empty MRMTransitionGroup with two dummy transitions

```
typedef MRMTransitionGroup<MSChromatogram, ReactionMonitoringTransition> TrGroup;
  TrGroup createTransitionGroup()
  {
    TrGroup tr_group;
    tr_group.addChromatogram(MSChromatogram(), “transition1”);
    tr_group.addTransition(ReactionMonitoringTransition(), “transition1”);
    tr_group.addChromatogram(MSChromatogram(), “transition2”);
    tr_group.addTransition(ReactionMonitoringTransition(), “transition2”);
    tr_group.setTransitionGroupID(“tr_peptideA”);
    return tr_group;
  }
```

Note how the identifiers of the chromatograms and the assay information (ReactionMonitoringTransition) are matched so that downstream algorithms can utilize the meta-information stored in the assays for data analysis.

2.8 Maps

Although raw data maps, peak maps and feature maps are conceptually very similar they are stored in different data types. For raw data and peak maps, the default container is MSExperiment, which is an array of MSSpectrum instances. In contrast to raw data and peak maps, feature maps are not a collection of one-dimensional spectra, but an array of two-dimensional feature instances. The main data structure for feature maps is called FeatureMap.
Although MSExperiment and FeatureMap differ in the data they store, they also have things in common. Both store metadata that is valid for the whole map, i.e. sample description and instrument description. This data is stored in the common base class ExperimentalSettings.

2.9 MSExperiment

MSExperiment contains ExperimentalSettings (metadata of the MS run) and a vector<MSSpectrum>. The one-dimensional spectrum MSSpectrum is derived from SpectrumSettings (metadata of a spectrum).

Example: Tutorial_MSExperiment.cpp

The following example creates a MSExperiment containing four MSSpectrum instances. We then iterate over RT range (2,3) and m/z range (603,802) and print the peak positions using an AreaIterator. Then we show how we iterate over all spectra and peaks. In the commented out part, we show how to load/store all spectra and associated metadata from/to an mzML file.

```
#include <OpenMS/CONCEPT/Types.h>
#include <OpenMS/KERNEL/MSExperiment.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  // create a peak map containing 4 dummy spectra and peaks
  MSExperiment exp;
  // The following examples creates a MSExperiment containing four MSSpectrum instances.
  for (Size i = 0; i < 4; ++i)
  {
    MSSpectrum spectrum;
    spectrum.setRT(i);
    spectrum.setMSLevel(1);
    for (float mz = 500.0; mz <= 900; mz += 100.0)
    {
      Peak1D peak;
      peak.setMZ(mz + i);
      spectrum.push_back(peak);
    }

    exp.addSpectrum(spectrum);
  }
  // Iteration over the RT range (2,3) and the m/z range (603,802) and print the peak positions.
  for (auto it = exp.areaBegin(2.0, 3.0, 603.0, 802.0); it != exp.areaEnd(); ++it)
  {
    cout « it.getRT() « " - " « it->getMZ() « endl;
  }
  // Iteration over all peaks in the experiment.
  // Output: RT, m/z, and intensity
  // Note that the retention time is stored in the spectrum (not in the peak object)
  for (auto s_it = exp.begin(); s_it != exp.end(); ++s_it)
  {
    for (auto p_it = s_it->begin(); p_it != s_it->end(); ++p_it)
    {
      cout « s_it->getRT() « " - " « p_it->getMZ() « " " « p_it->getIntensity() « endl;
    }
  }
  // We could store the spectra to a mzML file with:
  // MzMLFile mzml;
  // mzml.store(filename, exp);

  // And load it with
  // mzml.load(filename, exp);
  // If we wanted to load only the MS2 spectra we could speed up reading by setting:
  // mzml.getOptions().addMSLevel(2);
  // before executing: mzml.load(filename, exp);
  return 0;
} //end of main
```

2.10 FeatureMap

FeatureMap, the container for features, is simply a vector<Feature>. Additionally, it is derived from Experimental↩ Settings, to store the meta information. All peak and feature containers (MSSpectrum, MSExperiment, Feature↩ Map) are also derived from RangeManager. This class facilitates the handling of MS data ranges. It allows to calculate and store both the position range and the intensity range of the container.

Example: Tutorial_FeatureMap.cpp

The following examples creates a FeatureMap containing two Feature instances. Then we iterate over all features and output the retention time and m/z. We then show, how to use the underlying range manager to retrieve FeatureMap boundaries in rt, m/z, and intensity.

```
#include <OpenMS/KERNEL/FeatureMap.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
```

```
{
  // Insert of two features into a map and iterate over the features.
  FeatureMap map;
  Feature feature;
  feature.setRT(15.0);
  feature.setMZ(571.3);
  map.push_back(feature); //append feature 1
  feature.setRT(23.3);
  feature.setMZ(1311.3);
  map.push_back(feature); //append feature 2
  // Iteration over FeatureMap
  for (auto it = map.begin(); it != map.end(); ++it)
  {
    cout « it->getRT() « " - " « it->getMZ() « endl;
  }
  // Calculate and output the ranges
  map.updateRanges();
  cout « "Int: " « map.getMinInt() « " - " « map.getMaxInt() « endl;
  cout « "RT:  " « map.getMin()[0] « " - " « map.getMax()[0] « endl;
  cout « "m/z: " « map.getMin()[1] « " - " « map.getMax()[1] « endl;
  // ... and many more
  return 0;
} //end of main
```

2.11 File Formats

mzML	The HUPO-PSI standard format for mass spectrometry data
mzIdentML	The HUPO-PSI standard format for identification results data from any search engines
mzTAB	The HUPO-PSI standard format for reporting MS-based proteomics and metabolomics results
traML	The HUPO-PSI standard format for exchange and transmission lists for selected reaction monitoring (SRM) experiments
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 identification results
trafoXML	The OpenMS format for storing of transformations
OpenSWATH	

For further information of the HUPO Proteomics Standards Initiative please visit: http://www.psidev.info/

2.12 Logging

To make direct output to std::out and std::err more consistent, OpenMS provides several low-level macros:
LOG_FATAL_ERROR,
LOG_ERROR
LOG_WARN,
LOG_INFO and
LOG_DEBUG
which should be used instead of the less descriptive std::out and std::err streams.
If you are writing an OpenMS tool, you can also use the ProgressLogger to indicate how many percent of the processing has already been performed:

Example: Tutorial_Logger.cpp
Logging the Tool Progress

```
ProgressLogger progresslogger;
  progresslogger.setLogType(log_type_);  // set the log type (command line or a file)
  // set start progress (0) and end (ms_run.size() = the number of spectra)
  progresslogger.startProgress(0, ms_run.size(), "Doing some calculation...");
  for (PeakMap::Iterator it = ms_run.begin(); it != ms_run.end(); ++it)
  {
```

```
// update progress
     progresslogger.setProgress(ms_run.end() - it);

// do the actual calculations and processing ...
}
progresslogger.endProgress();
```

Depending on how the user configures the tool, this output is written to the command line or a log file.

2.13 Identifications

Identifications of proteins, peptides, and the mapping between peptides and proteins (or groups of proteins) are stored in dedicated data structures. These data structures are typically stored to disc as idXML or mzIdentML file. The highest-level structure is ProteinIdentification. It stores all identified proteins of an identification run as ProteinHit objects + additional metadata (search parameters, etc.). Each ProteinHit contains the actual Proteinaccession, an associated score, and (optionally) the protein sequence. A ProteinIdentification object stores the data corresponding to a single identified spectrum or feature. It has members for the retention time, m/z, and a vector of PeptideHits. Each PeptideHit stores the information of a specific peptide-to-spectrum match (e.g., the score and the peptide sequence). Each PeptideHit also contains a vector of PeptideEvidence objects which store the reference to one (or in the case the peptide maps to multiple proteins multiple) Proteins and the position therein.

Example: Tutorial_IdentificationClasses.cpp

Create all identification data needed to store an idXML file

```
#include <OpenMS/METADATA/PeptideIdentification.h>
#include <OpenMS/METADATA/ProteinIdentification.h>
#include <OpenMS/METADATA/PeptideHit.h>
#include <OpenMS/DATASTRUCTURES/String.h>
#include <OpenMS/CHEMISTRY/AASequence.h>
#include <OpenMS/FORMAT/IdXMLFile.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  // Create new protein identification object corresponding to a single search
  vector<ProteinIdentification> protein_ids;
  ProteinIdentification protein_id;
  protein_id.setIdentifier("Identifier");
  // Each ProteinIdentification object stores a vector of protein hits
  vector<ProteinHit> protein_hits;
  ProteinHit protein_hit = ProteinHit();
  protein_hit.setAccession("MyAccession");
  protein_hit.setSequence("PEPTIDEPEPTIDEPEPTIDEPEPTIDER");
  protein_hit.setScore(1.0);
  protein_hits.push_back(protein_hit);
  protein_id.setHits(protein_hits);
  DateTime now = DateTime::now();
  String date_string = now.getDate();
  protein_id.setDateTime(now);
  // Example of possible search parameters
  ProteinIdentification::SearchParameters search_parameters;
  search_parameters.db = "database";
  search_parameters.charges = "+2";
  protein_id.setSearchParameters(search_parameters);
  // Some search engine meta data
  protein_id.setSearchEngineVersion("v1.0.0");
  protein_id.setSearchEngine("SearchEngine");
  protein_id.setScoreType("HyperScore");
  protein_ids.push_back(protein_id);
  // Iterate over protein identifications and protein hits
  for (auto it = protein_ids.begin(); it != protein_ids.end(); ++it)
  {
    for (auto hit = it->getHits().begin(); hit < it->getHits().end(); ++hit)
    {
      cout « "Protein hit accession: " « hit->getAccession() « endl;
      cout « "Protein hit sequence: " « hit->getSequence() « endl;
      cout « "Protein hit score: " « hit->getScore() « endl;
    }
  }
  // Create new peptide identifications
  vector<PeptideIdentification> peptide_ids;
  PeptideIdentification peptide_id;
```

```
  peptide_id.setRT(1243.56);
  peptide_id.setMZ(440.0);
  peptide_id.setScoreType("ScoreType");
  peptide_id.setHigherScoreBetter(false);
  peptide_id.setIdentifier("Identifier");
  // define additional meta value for the peptide identification
  peptide_id.setMetaValue("AdditionalMetaValue", "Value");
  // add PeptideHit to a PeptideIdentification
  vector<PeptideHit> peptide_hits;
  PeptideHit peptide_hit;
  peptide_hit.setScore(1.0);
  peptide_hit.setRank(1);
  peptide_hit.setCharge(2);
  peptide_hit.setSequence(AASequence::fromString("DLQM(Oxidation)TQSPSSLSVSVGDR"));
  peptide_hits.push_back(peptide_hit);
  // add second best PeptideHit to the PeptideIdentification
  peptide_hit.setScore(1.5);
  peptide_hit.setRank(2);
  peptide_hit.setCharge(2);
  peptide_hit.setSequence(AASequence::fromString("QLDM(Oxidation)TQSPSSLSVSVGDR"));
  peptide_hits.push_back(peptide_hit);
  // add PeptideHit to PeptideIdentification
  peptide_id.setHits(peptide_hits);
  // add PeptideIdentification
  peptide_ids.push_back(peptide_id);
  // We could now store the identification data in an idXML file
  // IdXMLFile().store(outfile, protein_ids, peptide_ids);
  // And load it back with
  // IdXMLFile().load(outfile, protein_ids, peptide_ids);
  // Iterate over PeptideIdentification
  for (const auto& peptide_id : peptide_ids)
  {
    // Peptide identification values
    cout « "Peptide ID m/z: " « peptide_id.getMZ() « endl;
    cout « "Peptide ID rt: " « peptide_id.getRT() « endl;
    cout « "Peptide ID score type: " « peptide_id.getScoreType() « endl;
    // PeptideHits
    for (const auto& scored_hit : peptide_id.getHits())
    {
      cout « " - Peptide hit rank: " « scored_hit.getRank() « endl;
      cout « " - Peptide hit sequence: " « scored_hit.getSequence().toString() « endl;
      cout « " - Peptide hit score: " « scored_hit.getScore() « endl;
    }
  }
  // ...
  return 0;
}
```

2.14 Chemistry

2.15 Element, ElementDB, EmpiricalFormula

An Element object is the representation of an element. It can store the name, symbol and mass (average/mono) and natural abundances of isotopes. Elements are retrieved from the ElementDB singleton which is created from the file "/OpenMS/CHEMISTRY/Elements.xml". The EmpiricalFormula object can be used to represent the empirical formula of a compound as well as to extract its natural isotope abundance and weight.

Example: Tutorial_Element.cpp

Work with Element object

```
#include <OpenMS/CHEMISTRY/ElementDB.h>
#include <OpenMS/CHEMISTRY/Element.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
Int main()
{
  const ElementDB * db = ElementDB::getInstance();
  // extract carbon element from ElementDB
  // .getResidue("C") would work as well
  Element carbon = *db->getElement("Carbon");
  // output name, symbol, monoisotopic weight and average weight
  cout « carbon.getName() « " "
       « carbon.getSymbol() « " "
       « carbon.getMonoWeight() « " "
```

```
      « carbon.getAverageWeight() « endl;
  return 0;
} //end of main
```

Example: Tutorial_EmpiricalFormula.cpp

Extract isotope distribution and monoisotopic weight of an EmpiricalFormula object

```
#include <OpenMS/CHEMISTRY/EmpiricalFormula.h>
#include <OpenMS/CHEMISTRY/ElementDB.h>
#include <OpenMS/CHEMISTRY/ISOTOPEDISTRIBUTION/CoarseIsotopePatternGenerator.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
Int main()
{
  EmpiricalFormula methanol("CH3OH"), water("H2O");
  // sum up empirical formula
  EmpiricalFormula sum = methanol + water;
  // get element from ElementDB
  const Element * carbon = ElementDB::getInstance()->getElement("Carbon");
  // output number of carbon atoms and average weight
  cout « sum « " "
       « sum.getNumberOf(carbon) « " "
       « sum.getAverageWeight() « endl;
  // extract the isotope distribution
  IsotopeDistribution iso_dist = sum.getIsotopeDistribution(CoarseIsotopePatternGenerator(3));
  for (const auto& it : iso_dist)
  {
    cout « it.getMZ() « " " « it.getIntensity() « endl;
  }
  return 0;
} //end of main
```

2.16 AASequence - Representing a Peptide

An AASequence object stores a (potentially chemically modified) peptide. It can conveniently be constructed from the amino acid sequence (e.g., a string or a string literal "DEFIANGR"). Modifications may be encoded using the unimod name. Once constructed, many convenient functions are available to calculate peptide or ion properties.

Example: Tutorial_AASequence.cpp

Compute and output basic AASequence properties

```
#include <OpenMS/CHEMISTRY/AASequence.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  // generate AASequence object from String
  const String s = "DEFIANGER";
  AASequence peptide1 = AASequence::fromString(s);
  // generate AASequence object from string literal
  AASequence peptide2 = AASequence::fromString("PEPTIDER");
  // extract prefix and suffix
  AASequence prefix(peptide1.getPrefix(2));
  AASequence suffix(peptide1.getSuffix(3));
  cout « peptide1.toString() « " "
       « prefix « " "
       « suffix « endl;

  // create chemically modified peptide
  AASequence peptide_meth_ox = AASequence::fromString("PEPTIDESEKUEM(Oxidation)CER");
  cout « peptide_meth_ox.toString() « " "
       « peptide_meth_ox.toUnmodifiedString()
       « endl;
  // mass of the full, uncharged peptide
  double peptide_mass_mono = peptide_meth_ox.getMonoWeight();
  cout « "Monoisotopic mass of the uncharged, full peptide: " « peptide_mass_mono « endl;
  double peptide_mass_avg = peptide_meth_ox.getAverageWeight();
  cout « "Average mass of the uncharged, full peptide: " « peptide_mass_avg « endl;
  // mass of the 2+ charged b-ion with the given sequence
  double ion_mass_2plus = peptide_meth_ox.getMonoWeight(Residue::BIon, 2);
  cout « "Mass of the doubly positively charged b-ion: " « ion_mass_2plus « endl;
```

```
  // ... many more
  return 0;
}
```

Internally, an AASequence object is composed of Residues.

2.17 Residue, ResidueDB

Residues are the building blocks of AASequence objects. They store physico-chemical properties of specific amino acids. ResidueDB stores that data and is initialized from the file "data/CHEMISTRY/residues.xml".

Example: Tutorial_Residue.cpp

Compute and output basic Residue properties

```
#include <OpenMS/CHEMISTRY/ResidueDB.h>
#include <OpenMS/CHEMISTRY/Residue.h>
#include <OpenMS/CHEMISTRY/AASequence.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  // get ResidueDB singleton
  ResidueDB const * res_db = ResidueDB::getInstance();
  // query Lysine
  Residue const * lys = res_db->getResidue("Lysine");
  cout « lys->getName() « " "
       « lys->getThreeLetterCode() « " "
       « lys->getOneLetterCode() « " "
       « lys->getFormula().toString() « " "
       « lys->getAverageWeight() « " "
       « lys->getMonoWeight() « endl;
  // one letter code query of Arginine
  Residue const * arg = res_db->getResidue('R');
  cout « arg->getName() « " "
       « arg->getFormula().toString() « " "
       « arg->getMonoWeight() « endl;
  // construct a AASequence object, query a residue
  // and output some of its properties
  AASequence aas = AASequence::fromString("DEFIANGER");
  cout « aas[3].getName() « " "
       « aas[3].getFormula().toString() « " "
       « aas[3].getMonoWeight() « endl;
  return 0;
} //end of main
```

2.18 ResidueModification, ModificationsDB

If a residue is modified (e.g. phosphorylation of an amino acid) it can be stored in the ResidueModification class. The ResidueModification class stores information about chemical modifications of residues. Each Residue↩ Modification has an ID, the residue that can be modified with this modification and the difference in mass between the unmodified and the modified residue, among other information. The Residue class allows to set one modification per residue and the mass difference of the modification is accounted for in the mass of the residue. The class ModificationsDB is a database of ResidueModifications. These are mostly initialized from the file "/share/↩ CHEMISTRY/unimod.xml" containing a slightly modified version of the UniMod database of modifications. ModificationsDB has functions to search for modifications by name or mass.

Example: Tutorial_ResidueModification.cpp

Set a ResidueModification on a Residue

```
// --------------------------------------------------------------------------
//                   OpenMS -- Open-Source Mass Spectrometry
// --------------------------------------------------------------------------
// Copyright The OpenMS Team -- Eberhard Karls University Tuebingen,
// ETH Zurich, and Freie Universitaet Berlin 2002-2018.
//
// This software is released under a three-clause BSD license:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
```

```
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of any author or any participating institution
//    may be used to endorse or promote products derived from this software
//    without specific prior written permission.
// For a full list of authors, refer to the file AUTHORS.
// --------------------------------------------------------------------------
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
// ARE DISCLAIMED. IN NO EVENT SHALL ANY OF THE AUTHORS OR THE CONTRIBUTING
// INSTITUTIONS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
// OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
// WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
// OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
// ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
#include <OpenMS/CHEMISTRY/AASequence.h>
#include <OpenMS/CHEMISTRY/ResidueModification.h>
#include <OpenMS/CHEMISTRY/ModificationsDB.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  // construct a AASequence object, query a residue
  // and output some of its properties
  AASequence aas = AASequence::fromString("DECIANGER");
  cout « aas[2].getName() « " "
       « aas[2].getFormula().toString() « " "
       « aas[2].getModificationName() « " "
       « aas[2].getMonoWeight() « endl;

  // find a modification in ModificationsDB
  // and output some of its properties
  // getInstance() returns a pointer to a ModsDB instance
  ResidueModification mod = ModificationsDB::getInstance()->getModification("Carbamidomethyl (C)");
  cout « mod.getOrigin() « " "
       « mod.getFullId() « " "
       « mod.getDiffMonoMass() « " "
       « mod.getMonoMass() « endl;

  // set the modification on a residue of a peptide
  // and output some of its properties (the formula and mass have changed)
  // in this case ModificationsDB is used in the background
  // to relate the name of the mod to its attributes
  aas.setModification(2, "Carbamidomethyl (C)");
  cout « aas[2].getName() « " "
    « aas[2].getFormula().toString() « " "
    « aas[2].getModificationName() « " "
    « aas[2].getMonoWeight() « endl;
  return 0;
} //end of main
```

2.19 TheoreticalSpectrumGenerator

The TheoreticalSpectrumGenerator generates ion ladders from AASequences.

Example: Tutorial_TheoreticalSpectrumGenerator.cpp

Generate theoretical spectra

```
#include <OpenMS/CHEMISTRY/TheoreticalSpectrumGenerator.h>
#include <OpenMS/CHEMISTRY/AASequence.h>
#include <OpenMS/KERNEL/MSSpectrum.h>
#include <OpenMS/KERNEL/MSExperiment.h>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  // initialize a TheoreticalSpectrumGenerator
  TheoreticalSpectrumGenerator tsg;
  // get current parameters
  // in this case default parameters, since we have not changed any yet
```

```
  Param tsg_settings = tsg.getParameters();

  // with default parameters, only b- and y-ions are generated,
  // so we will add a-ions
  tsg_settings.setValue("add_a_ions", "true");

  // store ion types for each peak
  tsg_settings.setValue("add_metainfo", "true");

  // set the changed parameters for the TSG
  tsg.setParameters(tsg_settings);

  PeakSpectrum theoretical_spectrum;
  // initialize peptide to be fragmented
  AASequence peptide = AASequence::fromString("DEFIANGER");

  // generate a-, b- and y- ion spectrum of the peptide
  // with all fragment charges from 1 to 2
  tsg.getSpectrum(theoretical_spectrum, peptide, 1, 2);

  // output of masses and meta information (ion-types) of some peaks
  const PeakSpectrum::StringDataArray& ion_types = theoretical_spectrum.getStringDataArrays().at(0);
  cout « "Mass of second peak: " « theoretical_spectrum[1].getMZ()
       « " | Ion type of second peak: " « ion_types[1] « endl;
  cout « "Mass of tenth peak: " « theoretical_spectrum[9].getMZ()
       « " | Ion type of tenth peak: " « ion_types[9] « endl;

  return 0;
} //end of main
```

2.20 DigestionEnzymeProtein, ProteaseDB and ProteaseDigestion

OpenMS provides the most common digestion enzymes (DigestionEnzymeProtein) used in MS. They are stored in the ProteaseDB singleton and loaded from "/share/CHEMISTRY/Enzymes.xml".

Example: Tutorial_Enzyme.cpp

Digest amino acid sequence

```
#include <OpenMS/CHEMISTRY/AASequence.h>
#include <OpenMS/CHEMISTRY/ProteaseDigestion.h>
#include <vector>
#include <iostream>
using namespace OpenMS;
using namespace std;
int main()
{
  ProteaseDigestion protease;
  // in this example, we don't produce peptides with missed cleavages
  protease.setMissedCleavages(0);
  // output the number of tryptic peptides (no cut before proline)
  protease.setEnzyme("Trypsin");
  cout « protease.peptideCount(AASequence::fromString("ACKPDE")) « " "
       « protease.peptideCount(AASequence::fromString("ACRPDEKA"))
       « endl;
  // digest C-terminally amidated peptide
  vector<AASequence> products;
  protease.digest(AASequence::fromString("ARCDRE.(Amidated)"), products);
  // output digestion products
  for (const AASequence p : products)
  {
    cout « p.toString() « " ";
  }
  cout « endl;
  // allow many miss-cleavages
  protease.setMissedCleavages(10);
  protease.digest(AASequence::fromString("ARCDRE.(Amidated)"), products);
  // output digestion products
  for (const AASequence p : products)
  {
    cout « p.toString() « " ";
  }
  cout « endl;
  // ... many more
  return 0;
}
```

3 Tool development

3.1 TOPP-Tool

TOPP (The OpenMS Pipeline) tools are small command line applications built using the OpenMS library. They act as building blocks for complex analysis workflows and may perform e.g. simple signal processing tasks like filtering, up to more complex tasks like protein inference and quantitation over several MS runs. Common to all TOPP tools is a command line interface allowing automatic integration into workflow engines like KNIME. They are the preferred way to integrate novel methods as application into OpenMS. When we first create a novel TOPP tool it is considered unstable. To set it apart from the stable and well tested tools it gets first created as TOPP Util (note: the name "util" has historic reasons and may be changed to unstable tools in the future). If it is well tested it will be promoted to a stable Tool in future OpenMS versions.
Imagine that you want to create a new tool that allows filtering of sequence databases. What you usually would first do is check if such or similar functionality has already been implemented in any of the >150 TOPP tools. If you are unsure which one to use, just ask on the mailing list, the gitter chat or contact one of the developers directly. The following subsection demonstrates how the original "DatabaseFilter" tool was created from scratch an integrated into OpenMS. Basically any tool you want to integrate needs to follow the steps outlined below.
But let's first get started by defining what our tool should actually do: The DatabaseFilter tool should provide functionality to reduce a fasta database by filtering its entries based on different criteria. A simple criterion could be the length of a protein. To make the task a bit more interesting and to show other parts of the OpenMS library, we will start with a bit more complex filtering step that keeps all entries from the fasta database that have been identified in a peptide search (e.g., using X!Tandem, Mascot or MSGF+). This functionality might come in handy if the size of large databases needs to be reduced to a manageable size. In addition, we want the user to be able to choose between keeping and removing matching protein id.

3.2 Create and register a minimal tool in OpenMS

- Create an empty file src/utils/DatabaseFilter.cpp
- Add the scaffold code for a minimal TOPP tool. Text in bold will later be adapted to our DatabaseFilter tool.

Example: Tutorial_Template.cpp
Template for OpenMS tool development

```
// --------------------------------------------------------------------------
//                   OpenMS -- Open-Source Mass Spectrometry
// --------------------------------------------------------------------------
// Copyright The OpenMS Team -- Eberhard Karls University Tuebingen,
// ETH Zurich, and Freie Universitaet Berlin 2002-2018.
//
// This software is released under a three-clause BSD license:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of any author or any participating institution
//    may be used to endorse or promote products derived from this software
//    without specific prior written permission.
// For a full list of authors, refer to the file AUTHORS.
// --------------------------------------------------------------------------
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
// ARE DISCLAIMED. IN NO EVENT SHALL ANY OF THE AUTHORS OR THE CONTRIBUTING
// INSTITUTIONS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
// OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
// WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
// OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
// ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// --------------------------------------------------------------------------
// $Maintainer: Maintainer $
// $Authors: Author1, Author2 $
```

```
// --------------------------------------------------------------------------
#include <OpenMS/APPLICATIONS/TOPPBase.h>
using namespace OpenMS;
using namespace std;
//-------------------------------------------------------------
// Doxygen docu
//-------------------------------------------------------------
// We do not want this class to show up in the docu:

class TOPPNewTool :
  public TOPPBase
{
public:
  TOPPNewTool() :
    TOPPBase("NewTool", "Template for Tool creation", false)
  {
  }
protected:
  // this function will be used to register the tool parameters
  // it gets automatically called on tool execution
  void registerOptionsAndFlags_()
  {
  }
  // the main_ function is called after all parameters are read
  ExitCodes main_(int, const char **)
  {
    //-------------------------------------------------------------
    // parsing parameters
    //-------------------------------------------------------------
    //-------------------------------------------------------------
    // reading input
    //-------------------------------------------------------------
    //-------------------------------------------------------------
    // calculations
    //-------------------------------------------------------------
    //-------------------------------------------------------------
    // writing output
    //-------------------------------------------------------------
  }
};
// the actual main function needed to create an executable
int main(int argc, const char ** argv)
{
  TOPPNewTool tool;
  return tool.main(argc, argv);
}
```

- Now add a line with DatabaseFilter.cpp to src/utils/executables.cmake. This registers the novel tool in the OpenMS build system.
- Then add the tool to getUtilList() in src/openms/source/APPLICATIONS/ToolHandler.cpp This creates a manual (doxygen) page with the information –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 yourself as Maintainer/Author
- Write the basic documentation (doxygen docu). You probably need to refine it later but you can already insert the correct Toolname etc..

3.3 Define tool parameters

Define tool parameters Each TOPP tool defines a set of parameters that will be available from the command line, KNIME, and other workflow systems. This is done in the void registerOptionsAndFlags_() method. In our case we want to read a protein database (fasta format), a file containing identification data (idXML format), and an option to switch between keeping (whitelisting) and removing (blacklisting) entries based on the filter result. This is our input. The reduced database forms the output and should be written to a protein database in fasta format. This is easily done by adding following lines to:

Example: Tutorial_Final.cpp
Registration of tool parameters

```
void registerOptionsAndFlags_()
{
  registerInputFile_("in", "<file>", "","Input FASTA file, containing a database.");
  setValidFormats_("in", ListUtils::create<String>("fasta"));
  registerInputFile_("id", "<file>", "", "Input file containing identified peptides and proteins.");
  setValidFormats_("id", ListUtils::create<String>("idXML,mzid"));
  registerStringOption_("method", "<choice>", "whitelist", "Switch between white-/blacklisting of protein IDs", false);
  setValidStrings_("method", ListUtils::create<String>("whitelist,blacklist"));
  registerOutputFile_("out", "<file>", "", "Output FASTA file where the reduced database will be written to.");
  setValidFormats_("out", ListUtils::create<String>("fasta"));
}
```

Functions, classes and references can be checked in the OpenMS / TOPP documentation (ftp://ftp.mi.↩ fu-berlin.de/pub/OpenMS/release-documentation/html/index.html)

3.4 Read tool parameters

After a tool is executed, the registered parameters are available in the main_ function of the TOPP tool and can be read using the getStringOption_ method. Special methods for integers, lists and floating point parameters exist and are in the TOPPBase documentation but are not needed for this example.

Example: Tutorial_Final.cpp

```
//-------------------------------------------------------------
// parsing parameters
//-------------------------------------------------------------
String in(getStringOption_("in"));
String ids(getStringOption_("id"));
String method(getStringOption_("method"));
bool whitelist = (method == "whitelist");
String out(getStringOption_("out"));
```

3.5 Read Input Files

First the different file formats and data structures for peptide identifications have to be included at the top of the file.

Example: Tutorial_Final.cpp

Add essential includes

```
#include <OpenMS/FORMAT/FileHandler.h>
#include <OpenMS/FORMAT/IdXMLFile.h>
#include <OpenMS/FORMAT/FASTAFile.h>
#include <OpenMS/FORMAT/MzIdentMLFile.h>
#include <OpenMS/FORMAT/FileTypes.h>
#include <OpenMS/METADATA/PeptideIdentification.h>
#include <OpenMS/APPLICATIONS/TOPPBase.h>
```

Read the input files

```
vector<FASTAFile::FASTAEntry> db;
FASTAFile().load(in, db);
```

Note: both peptide_identifications and protein_identifications contain protein accessions. The difference between them is that protein_identifications only contain the inferred set of protein accessions while peptide_identifications contains all protein accessions the peptides map to. We consider only the larger set of protein accessions stored in the peptide identifications. In principle, it would be easy to add another parameter that adds a filter for the inferred accessions stored in protein_identifications.

3.6 Add the tool functionality

First, the accessions are extracted from the IdXML file. Here knowledge of the data structure is needed to extract the protein accessions. The class PeptideIdentification stores general information about a single identified spectrum (e.g., retention time, precursor mass-to-charge). A vector of PeptideHits is stored in each PeptideIdentification object and represent the potentially multiple PSMs of a single spectrum. They can be returned by calling .getHits(). Each peptide sequence stored in a PeptideHit may map to one or multiple proteins. This peptide to protein mapping

information is stored in a vector of PeptideEvidence accessible by .getPepitdeEvidences(). From each of these evidences we can extract the protein accession with .getProteinAccession().
To store all proteins accessions in the set id_accessions, we write:

Example: Tutorial_Final.cpp
Store protein accessions

```
void filterByProteinIDs_(const vector<FASTAFile::FASTAEntry>& db, const vector<PeptideIdentification>&
      peptide_identifications, bool whitelist, vector<FASTAFile::FASTAEntry>& db_new)
{
  set<String> id_accessions;
  for (Size i = 0; i != peptide_identifications.size(); ++i)
  {
    const PeptideIdentification& id = peptide_identifications[i];
    const vector<PeptideHit>& hits = id.getHits();
    for (Size k = 0; k != hits.size(); ++k)
    {
      const vector<PeptideEvidence>& evidences = hits[k].getPeptideEvidences();
      for (Size m = 0; m != evidences.size(); ++m)
      {
        const String& id_accession = evidences[m].getProteinAccession();
        id_accessions.insert(id_accession);
      }
    }
  }
```

Now that we assembled the set of all protein accessions we are ready to compare them to the fasta_accessions. If they are similar and the method whitelist or they are different and the method blacklist was chosen, the fasta entries are copied to the new fasta database.

Example: Tutorial_Final.cpp
Add method functionality

```
  for (Size i = 0; i != db.size() ; ++i)
  {
    const String& fasta_accession = db[i].identifier;
    const bool found = id_accessions.find(fasta_accession) != id_accessions.end();
    if ((found && whitelist) || (!found && !whitelist)) //either found in the whitelist or not found in the blacklist
    {
      db_new.push_back(db[i]);
    }
  }
```

3.7 Write Output Files

Example: Tutorial_Final.cpp
Write the output

```
  FASTAFile().store(out, db_new);
```

3.8 Adding TOPP tests

Testing your tools is essential and required to promote your experimental util to an official TOPP tool. It is not mandatory to provide a test for a util but appreciated. For this test a .fasta and a compatible .idXML file have to be added to /src/tests/topp/. Further the test procedure has to be added to CMakeLists.txt in the same folder.

Example: Tutorial_Test.cpp
Add tests

```
# DatabaseFilter test:
add_test("UTILS_DatabaseFilter_1" ${TOPP_BIN_PATH}/DatabaseFilter -test -in ${DATA_DIR_TOPP}/DatabaseFilter_1.fasta
      -accession ${DATA_DIR_TOPP}/DatabaseFilter_1.idXML -out DatabaseFilter_1_out.fasta.tmp)
add_test("UTILS_DatabaseFilter_1_out" ${DIFF} -in1 DatabaseFilter_1_out.fasta.tmp -in2
      ${DATA_DIR_TOPP}/DatabaseFilter_1_out.fasta )
set_tests_properties("UTILS_DatabaseFilter_1_out" PROPERTIES DEPENDS "UTILS_DatabaseFilter_1")
add_test("UTILS_DatabaseFilter_2" ${TOPP_BIN_PATH}/DatabaseFilter -test -in ${DATA_DIR_TOPP}/DatabaseFilter_1.fasta
      -accession ${DATA_DIR_TOPP}/DatabaseFilter_1.idXML -out DatabaseFilter_2_out.fasta.tmp -method blacklist)
add_test("UTILS_DatabaseFilter_2_out" ${DIFF} -in1 DatabaseFilter_2_out.fasta.tmp -in2
      ${DATA_DIR_TOPP}/DatabaseFilter_2_out.fasta )
set_tests_properties("UTILS_DatabaseFilter_2_out" PROPERTIES DEPENDS "UTILS_DatabaseFilter_2")
```

These tests run the program with the given parameters and then call a diff tool to compare the generated output to the expected output.

3.9 Finish documentation

We add it to the UTILS docu page (in doc/doxygen/public/UTILS.doxygen). Later (when we have a working application) we will write an application test (this is optional but recommended for Utils. For Tools it is mandatory). See TOPP tools above and add the test to the bottom of src/tests/topp/CMakeLists.txt.

3.10 Polish your code

This is how a util should look after code polishing: Here, the support for different formats was extended (idXML and MZIdentML). Since different filter criteria may be introduced in the future, the structure was slightly changed with a function for the filtering by ID (filterByProteinIDs_) - in order to allow higher flexibility when adding new a functionality later on.

Example: Tutorial_final.cpp
Polish your code - add additional functionality

```
// --------------------------------------------------------------------------
//                   OpenMS -- Open-Source Mass Spectrometry
// --------------------------------------------------------------------------
// Copyright The OpenMS Team -- Eberhard Karls University Tuebingen,
// ETH Zurich, and Freie Universitaet Berlin 2002-2018.
//
// This software is released under a three-clause BSD license:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of any author or any participating institution
//    may be used to endorse or promote products derived from this software
//    without specific prior written permission.
// For a full list of authors, refer to the file AUTHORS.
// --------------------------------------------------------------------------
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
// ARE DISCLAIMED. IN NO EVENT SHALL ANY OF THE AUTHORS OR THE CONTRIBUTING
// INSTITUTIONS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
// OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
// WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
// OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
// ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// --------------------------------------------------------------------------
// $Maintainer: Oliver Alka $
// $Authors: Oliver Alka $
// --------------------------------------------------------------------------
#include <OpenMS/FORMAT/FileHandler.h>
#include <OpenMS/FORMAT/IdXMLFile.h>
#include <OpenMS/FORMAT/FASTAFile.h>
#include <OpenMS/FORMAT/MzIdentMLFile.h>
#include <OpenMS/FORMAT/FileTypes.h>
#include <OpenMS/METADATA/PeptideIdentification.h>
#include <OpenMS/APPLICATIONS/TOPPBase.h>
using namespace OpenMS;
using namespace std;
//-------------------------------------------------------------
//Doxygen docu
//-------------------------------------------------------------
// We do not want this class to show up in the docu:

class TOPPDatabaseFilter :
 public TOPPBase
{
public:
 TOPPDatabaseFilter() :
   TOPPBase("DatabaseFilter", "Filters a protein database (FASTA format) based on identified proteins", false)
```

```
 {
 }
protected:
 void registerOptionsAndFlags_()
 {
   registerInputFile_("in", "<file>", "","Input FASTA file, containing a database.");
   setValidFormats_("in", ListUtils::create<String>("fasta"));
   registerInputFile_("id", "<file>", "", "Input file containing identified peptides and proteins.");
   setValidFormats_("id", ListUtils::create<String>("idXML,mzid"));
   registerStringOption_("method", "<choice>", "whitelist", "Switch between white-/blacklisting of protein IDs", false);
   setValidStrings_("method", ListUtils::create<String>("whitelist,blacklist"));
   registerOutputFile_("out", "<file>", "", "Output FASTA file where the reduced database will be written to.");
   setValidFormats_("out", ListUtils::create<String>("fasta"));
 }
 void filterByProteinIDs_(const vector<FASTAFile::FASTAEntry>& db, const vector<PeptideIdentification>&
       peptide_identifications, bool whitelist, vector<FASTAFile::FASTAEntry>& db_new)
 {
   set<String> id_accessions;
   for (Size i = 0; i != peptide_identifications.size(); ++i)
   {
     const PeptideIdentification& id = peptide_identifications[i];
     const vector<PeptideHit>& hits = id.getHits();
     for (Size k = 0; k != hits.size(); ++k)
     {
       const vector<PeptideEvidence>& evidences = hits[k].getPeptideEvidences();
       for (Size m = 0; m != evidences.size(); ++m)
       {
         const String& id_accession = evidences[m].getProteinAccession();
         id_accessions.insert(id_accession);
       }
     }
   }
   LOG_INFO « "Protein IDs: " « id_accessions.size() « endl;
   for (Size i = 0; i != db.size() ; ++i)
   {
     const String& fasta_accession = db[i].identifier;
     const bool found = id_accessions.find(fasta_accession) != id_accessions.end();
     if ((found && whitelist) || (!found && !whitelist)) //either found in the whitelist or not found in the blacklist
     {
       db_new.push_back(db[i]);
     }
   }
 }
 ExitCodes main_(int, const char **)
 {
   //-------------------------------------------------------------
   // parsing parameters
   //-------------------------------------------------------------
   String in(getStringOption_("in"));
   String ids(getStringOption_("id"));
   String method(getStringOption_("method"));
   bool whitelist = (method == "whitelist");
   String out(getStringOption_("out"));
   //-------------------------------------------------------------
   // reading input
   //-------------------------------------------------------------
   vector<FASTAFile::FASTAEntry> db;
   FASTAFile().load(in, db);
   // Check if no filter criteria was given
   // If you add a new filter please check if it was set here as well
   if (ids.empty())
   {
     FASTAFile().store(out, db);
   }
   vector<FASTAFile::FASTAEntry> db_new;
   if (!ids.empty()) // filter by protein IDs
   {
     FileHandler fh;
     FileTypes::Type ids_type = fh.getType(ids);
     vector<ProteinIdentification> protein_identifications;
     vector<PeptideIdentification> peptide_identifications;
     if (ids_type == FileTypes::IDXML)
     {
       IdXMLFile().load(ids, protein_identifications, peptide_identifications);
     }
     else if (ids_type == FileTypes::MZIDENTML)
     {
       MzIdentMLFile().load(ids, protein_identifications, peptide_identifications);
```

```
    }
    else
    {
      writeLog_("Error: Unknown input file type given. Aborting!");
      printUsage_();
      return ILLEGAL_PARAMETERS;
    }
    LOG_INFO « "Identifications: " « ids.size() « endl;
    // run filter
    filterByProteinIDs_(db, peptide_identifications, whitelist, db_new);
  }
  //-------------------------------------------------------------
  // writing output
  //-------------------------------------------------------------
  LOG_INFO « "Database entries (before / after): " « db.size() « " / " « db_new.size() « endl;
  FASTAFile().store(out, db_new);
  return EXECUTION_OK;
 }
};
int main(int argc, const char ** argv)
{
 TOPPDatabaseFilter tool;
 return tool.main(argc, argv);
}
```

3.11 Open a pull request

Afterwards you can commit your changes to a new branch "feature/DatabaseFilter" of your OpenMS clone on github and submit a pull request on your github page. After a short review process by the OpenMS Team, the tool will be added the OpenMS Library.

4 Appendix

4.1 D-dimensional data points

The d-dimensional data points are needed in special cases only, e.g. in template classes that operate in any number of dimensions. The base class of the d-dimensional data points is DPeak. The methods to access the position are getPosition and setPosition. Note that the one-dimensional and two-dimensional data points also have the methods getPosition and setPosition. They are needed in order to be able to write algorithms that can operate on all data point types. It is, however, recommended not to use these members unless you really write such a generic algorithm.

4.2 OpenMS as external project

If OpenMS TOPP_tools and UTILS_tools are not sufficient for a certain scenario, you can either request changes to OpenMS or modify/extend your own fork of OpenMS. A third alternative is using OpenMS as a dependency while not touching OpenMS itself. Once you've finished your new tool, and it runs on the development machine, you're done. If you want to develop with OpenMS as external project have a look the example code (/share/Open↩MS/examples/external_code/).