BASTet: Berkeley Analysis and Storage Toolkit

Release devel

Oliver Rübel and Ben Bowen

May 24, 2016
## CONTENTS

1 Introduction .............................................. 3

2 Converting Files at NERSC and Making them Accessible ............... 5
   2.1 Making a converted file accessible to OpenMSI (Private) .......... 5
   2.2 Changing file permissions ........................................ 5
   2.3 convertToOMSI: Usage and Options ............................... 5

3 Integrating new file formats ................................ 9
   3.1 Developing a file reader ............................................ 9
   3.2 Integrating the file reader with OpenMSI ......................... 10

4 Developing a new Analysis for BASTet ............................. 13
   4.1 Wrapping a function: The quick-and-dirty way .................... 13
       4.1.1 Option 1: Explicitly track specific executions of a function . 13
       4.1.2 Option 2: Implicitly track the last execution of a function . 14
       4.1.3 Example 1: Defining and using wrapped functions ............. 14
   4.2 Writing a new analysis class for BASTet – Part 1: Overview ........ 16
       4.2.1 Some important features of analysis_base .................... 17
           Slicing ..................................................................... 17
           Important Member Variables ........................................ 17
           I/O functions .......................................................... 18
           Web API Functions .................................................... 18
           Executing, saving, and restoring an analysis object ............. 19
   4.3 Writing a new analysis class for BASTet – Part 2: Analysis Template ........................................ 20
       4.3.1 Step 1) Basic integration ....................................... 20
           1.1 Creating a new analysis skeleton ................................ 21
           1.2 Specifying analysis inputs and outputs ......................... 22
           1.3: Implementing the execute_analysis function .................. 23
       4.3.2 Step 2) Integrating the Analysis with the OpenMSI Web API: 23
           2.1 Implementing the qslice pattern ................................ 24
           2.2 Implementing the qspectrum pattern ............................. 25
           2.3 Implementing the qmz pattern .................................... 26
       4.3.3 Step 3) Making your analysis self-sufficient ..................... 27
   4.4 Writing a new analysis class for BASTet – Part 3: Customizing core features ........................................... 29
       4.4.1 Custom data save ................................................... 29
       4.4.2 Custom analysis restore .......................................... 30
       4.4.3 Custom analysis execution ........................................ 30
           Customizing setting of parameters .................................. 30
           Customizing setting of default settings ............................ 30
           Customizing the recording of runtime information ............... 30
           Customizing the analysis execution ................................ 31
10.1 Subpackages .................................................................................. 77
  10.1.1 analysis Package ....................................................................... 78
    analysis Package ........................................................................... 78
    base Module .................................................................................. 96
    generic Module ........................................................................... 106
    analysis_views Module .................................................................. 108
    Subpackages ................................................................................ 110
      findpeaks Package ...................................................................... 110
        findpeaks Package .................................................................. 110
        omsi_findpeaks_global Module .............................................. 112
        omsi_findpeaks_local Module ................................................. 112
      findpeaks.third_party Package .................................................. 113
        findpeaks.third_party Package ................................................ 113
        findpeaks Module .................................................................. 113
      findpeaks.experimental Package ................................................. 113
        findpeaks.experimental Package .............................................. 113
        omsi_peakcube Module ............................................................. 114
        pfrun Module .......................................................................... 114
        omsi_lpf Module .................................................................... 114
        omsi_npg Module ................................................................... 116
        mypeakfinder Module .............................................................. 117
      multivariate_stats Package ......................................................... 117
        multivariate_stats Package ....................................................... 117
        omsi_nmf Module .................................................................... 120
      multivariate_stats.third_party Package ....................................... 120
        multivariate_stats.third_party Package .................................. 120
        nmf Module ............................................................................ 121
      multivariate_stats.experimental Package .................................... 121
        multivariate_stats.experimental Package ................................ 121
      msi_filtering Package ................................................................. 121
        msi_filtering Package ............................................................ 121
        omsi_tic_norm Module .............................................................. 124
      msi_filtering.third_party Package ................................................. 126
        msi_filtering.third_party Package ............................................ 126
      msi_filtering.experimental Package .............................................. 126
        msi_filtering.experimental Package ........................................ 126
        omsi_filter_by_mask Module .................................................... 126
        omsi_mask_by_cluster Module .................................................. 128
      compound_stats Package ............................................................. 130
        compound_stats Package ........................................................ 130
        omsi_score_midas Module ........................................................ 130
        omsi_score_pactolus Module ..................................................... 131
      compound_stats.third_party Package .......................................... 131
        compound_stats.third_party Package ....................................... 131
      compound_stats.experimental Package ....................................... 131
        compound_stats.experimental Package ................................... 131
  10.1.2 dataformat Package ................................................................... 132
    OMSI Dataformat Package .............................................................. 132
      omsi_file Package ..................................................................... 132
      format Module .......................................................................... 133
      common Module ........................................................................ 137
      main_file Module ..................................................................... 140
      experiment Module ................................................................... 141
      metadata_collection Module ..................................................... 143
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>instrument</td>
<td>145</td>
</tr>
<tr>
<td>methods</td>
<td>147</td>
</tr>
<tr>
<td>msidata</td>
<td>148</td>
</tr>
<tr>
<td>analysis</td>
<td>154</td>
</tr>
<tr>
<td>dependencies</td>
<td>158</td>
</tr>
<tr>
<td>file_reader_base</td>
<td>165</td>
</tr>
<tr>
<td>img_file</td>
<td>169</td>
</tr>
<tr>
<td>mzml_file</td>
<td>170</td>
</tr>
<tr>
<td>brukerflex_file</td>
<td>172</td>
</tr>
<tr>
<td>10.1.3 datastuctures Package</td>
<td>176</td>
</tr>
<tr>
<td>datastructures</td>
<td>176</td>
</tr>
<tr>
<td>analysis_data</td>
<td>176</td>
</tr>
<tr>
<td>dependency_data</td>
<td>180</td>
</tr>
<tr>
<td>run_info_data</td>
<td>182</td>
</tr>
<tr>
<td>Subpackages</td>
<td>184</td>
</tr>
<tr>
<td>metadata</td>
<td>184</td>
</tr>
<tr>
<td>metadata_data</td>
<td>184</td>
</tr>
<tr>
<td>metadata_ontologies</td>
<td>185</td>
</tr>
<tr>
<td>10.1.4 shared Package</td>
<td>185</td>
</tr>
<tr>
<td>data_selection</td>
<td>186</td>
</tr>
<tr>
<td>omsi_web_helper</td>
<td>196</td>
</tr>
<tr>
<td>spectrum_layout</td>
<td>198</td>
</tr>
<tr>
<td>log</td>
<td>199</td>
</tr>
<tr>
<td>mpi_helper</td>
<td>203</td>
</tr>
<tr>
<td>10.1.5 workflow Package</td>
<td>205</td>
</tr>
<tr>
<td>workflow</td>
<td>206</td>
</tr>
<tr>
<td>common</td>
<td>206</td>
</tr>
<tr>
<td>Subpackages</td>
<td>208</td>
</tr>
<tr>
<td>driver</td>
<td>208</td>
</tr>
<tr>
<td>base</td>
<td>209</td>
</tr>
<tr>
<td>cl_analysis_driver</td>
<td>209</td>
</tr>
<tr>
<td>cl_workflow_driver</td>
<td>212</td>
</tr>
<tr>
<td>executor</td>
<td>215</td>
</tr>
<tr>
<td>executor</td>
<td>215</td>
</tr>
<tr>
<td>base</td>
<td>215</td>
</tr>
<tr>
<td>greedy_executor</td>
<td>217</td>
</tr>
<tr>
<td>10.1.6 tools Package</td>
<td>218</td>
</tr>
<tr>
<td>convertToOMSI</td>
<td>218</td>
</tr>
<tr>
<td>run_analysis</td>
<td>222</td>
</tr>
<tr>
<td>run_workflow</td>
<td>222</td>
</tr>
<tr>
<td>Subpackages</td>
<td>222</td>
</tr>
<tr>
<td>tools.experimental</td>
<td>222</td>
</tr>
<tr>
<td>tools.experimental</td>
<td>222</td>
</tr>
<tr>
<td>tools.misc</td>
<td>222</td>
</tr>
<tr>
<td>tools.misc</td>
<td>222</td>
</tr>
<tr>
<td>make_thumb</td>
<td>222</td>
</tr>
<tr>
<td>create_peak_cube_overview</td>
<td>222</td>
</tr>
<tr>
<td>10.1.7 examples Package</td>
<td>222</td>
</tr>
<tr>
<td>simple_viewer</td>
<td>222</td>
</tr>
<tr>
<td>testHDF5Optimization</td>
<td>223</td>
</tr>
<tr>
<td>testHDF5Optimization_alignedWrite</td>
<td>223</td>
</tr>
<tr>
<td>test_multiprocess_slice</td>
<td>223</td>
</tr>
</tbody>
</table>
test_par Module ......................................................... 224
test_peakcube Module ............................................... 224
testhdf5_file_read Module ........................................... 224

10.1.8 templates Package ............................................. 224
templates Package ..................................................... 224
analysis_template Module ............................................. 224
BASTet has been originally developed as the analysis backend for the [OpenMSI](https://openmsi.nersc.gov/) science gateway.

A central goal of BASTet is to facilitate shareable and reproducible analysis and to bridge the gaps between the various stages in the typical life-cycle of new analysis methods, e.g., when transition from a research prototype to production or when integrating an analysis into more complex workflows or when sharing analysis results. BASTet provides users with an environment that makes it easy to develop and deploy analyses via OpenMSI by providing:

- standardized analysis interfaces that make analyses easily accessible to users and enable developers to build an accessible ecosystem of domain analytics,
- automatic provenance for reproducible analytics,
- standardized interfaces and a common format for storage, access, sharing and reuse of analysis results and raw MSI data
- support for integration of analyses to complex workflows, providing users a lightweight and easy-to-use entry-point to define and manage analysis workflows, and
- integrated tools and analytics to facilitate development and deployment.

While the design of BASTet has been motivated by the needs of OpenMSI, its core design and functionality are much more broadly applicable to other applications as well. Basket is implemented in Python using NumPy for data processing, h5py for HDF5-based data storage, and mpi4py for distributed parallel data processing.
To convert a mass spectrometry imaging file, e.g., in img or bruckerflex format, to HDF5 do the following:

```bash
ssh cori.nersc.gov
cd /project/projectdirs/openmsi/devel/convert
source setupEnvironment.csh
python convertToOMSI.py <infile1> <output HDF5File>
```

Note if you use the bash shell then use `setupEnvironment.bash` instead.

Note, if you want to use the output file from openmsi.nersc.gov then the output file path should be:

```
/project/projectdirs/openmsi/omsi_data_private/<username>/<filename>
```

where username is the name of the primary user that owns the file.

### 2.1 Making a converted file accessible to OpenMSI (Private)

The conversion script will by default automatically try to register new files with the OpenMSI site and assign them private to a single user if the output file is placed in the OpenMSI private data location:

```bash
python convertToOMSI.py <infile1> /project/projectdirs/openmsi/omsi_data_private/<username>/<filename>
```

The username in the path will determine the user the file is assigned to. E.g: The filename will also be the name used in the listing on the site. In order to generate the HDF5 file without adding to the database, use the `--no-add-to-db` command line option, e.g.:

### 2.2 Changing file permissions:

Using the OpenMSI website the owner of the file can assign permissions to files online:

[https://openmsi.nersc.gov/openmsi/resources/filemanager?file=<filename>](https://openmsi.nersc.gov/openmsi/resources/filemanager?file=<filename>)

### 2.3 convertToOMSI: Usage and Options

NOTE: In order to view a current, complete list of conversions options use:

```bash
python convertToOMSI.py --help
```
python convertToOMSI.py --help
USAGE: Call "convertToOMSI [options] imgBaseFile1 imgBaseFile2 ... imgBaseFileN HDF5File"

This converter script takes the basename (i.e., path+basefilename) of a single or multiple MSI files as input and converts them to HDF5. Each MSI file is stored as a separate experiment in the output HDF5 file. If an input file defines multiple regions, then those regions can either be stored as separate datasets of the same experiment and/or merged to a single MSI dataset. Using the various parameter settings described below, one can define how the conversion should be performed, how the data should be stored in HDF5, and indicate which analyses should be executed.

===HELPER OPTIONS===
--suggest-chunking: Iterate over all given input files and suggest a chunking strategy.
No data is converted when this option is given, i.e., no name for the HDF5File should be given, but only input files should be listed.

===ERROR HANDLING OPTIONS===
--error-handling <options>: Define how errors should be handled. Options are:
  i) terminate-and-cleanup (default): Terminate the conversion, delete the HDF5 file and do not add the file to the database.
  ii) terminate-only, : Leave the generated HDF5 output file in place but do not add the file to the database.
  iii) continue-on-error: Ignore errors if possible and continue, even if this means that some data may be missing from the output.
--email <email1 email2 ...>: Send notification in case of both error or success to the email address.
--email-success <email1 email2 ...>: Send notification in case of success to the given email address.
--email-error <email1 email2 ...>: Send notification in case of error to the given email address.

===INPUT DATA OPTIONS===
Default input data options: --format auto --regions split+merge
--format <option>: Define which file format is used as input. By default the program tries to automatically determine the input format. This option can be used to indicate the format explicitly to in case the auto option fails. Available options are:
  {'imzml_file': <class 'omsi.dataformat.imzml_file.imzml_file'>, 'bruckerflex_file': <class 'omsi.dataformat.bruckerflex_file.bruckerflex_file'>, ... 'img_file': <class 'omsi.dataformat.img_file.img_file'>, 'mzml_file': <class 'omsi.dataformat.mzml_file.mzml_file'>}
--regions <option>: Some file formats (e.g., brucker) allow multiple regions to be imaged and stored in a single file. This option allows one to specify how these regions should be treated during file conversion. E.g., one may want to store i) each region as a separate dataset in the output file (--regions split), ii) all regions combined in a single dataset (--regions merge), or both (--regions split+merge)
Available options are:
  ['split', 'merge', 'split+merge']

===FILE WRITE OPTIONS===

---FILE WRITE OPTIONS: Chunking---
Default HDF5 Chunking options: Enabled by default using --auto-chunking:
--auto-chunking: Automatically decide which chunking should be used. This option automatically generates two copies of the data, one with a chunking optimized for selection of spectra and another one optimized for selection of ion image slices. All --chunking, --no-chunking, and --optimized-chunking options are ignored if this parameter is given.
--chunking <x y z>: Use chunking when writing the HDF5 file. (DEFAULT, x=4, y=4, z=2048)
--no-chunking: Disable chunking when writing the HDF5 file. Use in combination with --no-compression since compression depends on chunking and will enable it if compression is used.
--optimized-chunking <x y z> : Use this option to generate additional copies of the data with different chunked data layouts. Generating multiple copies of the data with different chunked data layouts can help accelerate selective data read operations. (DEFAULT OFF). We recommend a spectra-aligned chunking for the raw data, e.g., '--chunking 1 1 32768' and an image-aligned chunked secondary copy of the data, e.g., '--optimized-chunking 20 20 100'.

---FILE WRITE OPTIONS: Compression---
HDF5 Compression: Default ON using (gzip, 4):
--compression: Enable compression using (gzip,4). NOTE: Compression requires the use of chunking.
--no-compression: Disable the use of compression.

===I/O OPTIONS===
--io <option>: Available options are: ['chunk', 'spectrum', 'all']
i) all : Read the full data in memory and write it at once
ii) spectrum : Read one spectrum at a time and write it to the file.
iii) chunk : Read one chunk at a time and write it to the file.
The io option applies only for the generation of subsequent chunkings and not the initial iteration over the file to generate the first convert.
iv) spectrum-to-image: Default option when creating image chunk version from a spectrum-chunk MSI dataset. Read a block of spectra at a time to complete a set of images and then write the block of images at once.
--io-block-limit <MB>: When using spectrum-to-image io (default when using auto-chunking), what should the maximum block in MB that we load into memory. (Default=2000MB)

===DATABASE OPTIONS===
These options control whether the generated output file should be added to a server database to manage web file access permissions
Default options are: --add-to-db --db-server http://openmsi.nersc.gov
--add-to-db : Explicitly add the output HDF5 file to the database. This option has no effect if --jobid is set as the file is added through the update of the job status in this case.
--no-add-to-db : Disable adding the file to the database.
--db-server : Specify the online server where the file should be registers. Default is http://openmsi.nersc.gov
--user : Name of the user that should be assigned as user. By default the user is determined automatically based on the file path.
--jobid : ID of the job. If set to 'auto' then the environment variable PBS_JOBID is used. NOTE: If job ID is set then we assume that the job has been scheduled via the automated system and that the job is managed. As such the file will be added, to the database by updating the job status and NOT by explicitly adding the file.

===ANALYSIS OPTIONS===
NMF: Default ON: (nc=20, timeout=600, niter=2000, tolerance=0.0001, raw=False)
--nmf : Compute the nmf for all the input data files and store the results in the HDF5 file. NOTE: If global peak-finding (fpg) is performed, then nmf will be performed on the peak-cube, otherwise on the raw data
--no-nmf: Disable the execution of nmf
--nmf-nc <number>: Number of components to be computed by the NMF. (default nc=20)
--nmf-timeout <number>: Maximum time in seconds to be used for computing the NMF. (default timeout=600)
--nmf-niter <number>: Number of iterations (minimum is 2)(default niter=2000)
--nmf-tolerance <number>: Tolerance value for a relative stopping condition. (default tolerance=0.0001)
--nmf-raw <number>: Force execution of the NMF on the raw data. By default the results from the global peak finding (--fpg) are used to compute the NMF.
--fpg : Compute the global peak finding for all input data files and save results in the HDF5 file (DEFAULT)
--no-fpg: Disable the global peak finding

Local Peak Finding: Default OFF:
--fpl : Compute the local peak finding for all input data files and save results in the HDF5 file
--no-fpl: Disable the local peak finding (DEFAULT)

TIC normalization:
--ticnorm : Compute tic normalization
--no-ticnorm : Disable computation of tic normalization (DEFAULT)

---OTHER OPTIONS---

Generate Thumbnail image: Default OFF:
--thumbnail: Generate thumbnail image for the file based on, in order of availability:
  * The first three components of the NMF
  * The three most intense peaks from the global peak finding (fpg)
  * The three most intense peaks in the raw data that are at least 1 percent of the total m/z range apart.
--no-thumbnail: Do not generate a thumbnail image.

Generate XDMF header file for output file: Default OFF:
--xdmf: Write XDMF XML-based header-file for the output HDF5 file.
--no-xdmf: Do not generate a XDMF XML-based header for the HDF5 file.

===Metadata Options===

NOTE: Input datasets are numbers starting from 0 based on there order on the command line.

--methods : JSON describing the experimental methods
--methods# : JSON describing the experimental methods for input file number #
--instrument : JSON dictionary describing the instrument
--instrument# : JSON dictionary describing the instrument for input file number #
--notes : JSON dictionary with additional user notes about the data
--notes# : JSON dictionary with additional notes for input file number #
3.1 Developing a file reader

In order to develop a new file reader we need to implement a corresponding class for the file format that inherits from `file_reader_base` (for formats that always contain a single region) or `file_reader_base_with_regions` (for formats that support multiple imaging regions in a single file). Both base classes are available in the `omsi.dataformat.file_reader_base` module. The developer then needs to implement the following functions:

```python
from omsi.dataformat.file_reader_base import file_reader_base

class formatname_file(file_reader_base):

    # 1. Implement the init function which must accept basename and requires_slicing as inputs
    def __init__(self, basename=None, requires_slicing=True):
        
        basename : Name of the file/folder to be opened
        requires_slicing: Boolean indicating whether the user requires array slicing via
        the __getitem__ function to work or not. This is an optimization, because many MSI
        data formats do not easily support arbitrary slicing of data but rather only
        iteration over spectra.
        
        super(formatname_file, self).__init__(basename, requires_slicing)  # 1.1 Call super __init__

        self.data_type = 'uint16'  # 1.2 Define the data type used
        self.shape = [0, 0, 0]  # 1.3 Define the shape of the dataset
        self.mz = 0  # 1.4 Define the m/z axis

    # 2. Implement __getitem__
    def __getitem__(self, key):
        # Implement array-based slicing for the format so that the data can be read
        # via [x,y,z]

    # 3. Implement close_file
    def close_file(self):
        # Function called to close any open files when deleting the object

    # 4. Implement is_valid_dataset
    @classmethod
    def is_valid_dataset(cls, name):
        # Given the name of a file or directory, determine whether the given
        # data defines a valid dataset for the current format
```

For file formats that support multiple regions the implementation is aside from a few additions the same. The main difference are:
1. Inherit from `file_reader_base_with_regions` instead of `file_reader_base`
2. Set the `self.region_dicts` and `self.select_region` attributes in the `__init__` function
3. Implement the `set_region_selection` function

```python
from omsi.dataformat.file_reader_base import file_reader_base_with_regions

class formatname_file(file_reader_base):
    # 1. Implement the init function which must accept basename and requires_slicing as inputs
    def __init__(self, basename=None, requires_slicing=True):
        super(formatname_file, self).__init__(basename, requires_slicing)
        # 1.1 Call super __init__
        self.data_type = 'uint16'
        # 1.2 Define the data type used
        self.shape = [0, 0, 0]
        # 1.3 Define the shape of the dataset
        self.mz = 0
        # 1.4 Define the m/z axis
        self.region_dicts = []
        # 1.5 Define a list of dicts where each dict describes
        # the parameters of a region. In the simplest case
        # of rectangular regions, a dict would specify the
        # 'origin' and 'extends' of a region.
        self.select_region = None
        # 1.6 Define the index of the selected region.
        # Set to None to treat dataset as a whole (i.e.,
        # merge all regions). If set to a region index,
        # then the __getitem__ function is expected to
        # behave as if the file consisted of just the
        # selected region and the self.shape parameter
        # must be set accordingly

    # 2-4: Implement the other functions of file_reader_base as described above.
    # Note, __getitem__ must consider the value of self.select_region and
    # treat any data requests as if they referred to the selected region
    # only. Depending on the data format this may require transformation
    # of the selection keys to locate the appropriate data

    # 5. Implement the set_region_selection function to allow a user to select a region
    def set_region_selection(self, region_index=None):
        """Define which region should be selected for local data reads.

        :param region_index: The index of the region that should be read. The shape of the
data will be adjusted accordingly. Set to None to select all regions and
treat the data as a single full 3D image.
        """

        # 5.1 Select all data
        if region_index is None:
            self.select_region = None
            # 5.1.1 Set the region selection to None
            self.shape = self.full_shape
            # 5.1.2 Define shape of the complete data

        # 5.2 Select a particular region
        elif region_index < self.get_number_of_regions():
            self.select_region = region_index
            # 5.2.1 Set region index
            self.shape = ...
            # 5.2.2 Define the 3D shape of the region
```

3.2 Integrating the file reader with OpenMSI

Integrating a new file reader with OpenMSI is simple:

1. Add the file reader module to the `omsi.dataformat` format module and
2. Add the name of your module to the `__all__` variable in `omsi.dataformat.__init__.py`
Once these steps are complete, the `omsi.dataformat.file_reader_base` module will automatically detect the new format and make it available as part of the file conversion script `omsi.tools.convertToOMSI`. To check which formats are registered, simply do:

```python
>>> from omsi.dataformat import *
>>> formats = file_reader_base.file_reader_base.available_formats()
>>> print formats
{'bruckerflex_file': <class 'omsi.dataformat.bruckerflex_file.bruckerflex_file'>,
 'img_file': <class 'omsi.dataformat.img_file.img_file'>}
```

Using this basic feature makes it possible to easily iterate over all available formats and the consistent interface described by the `file_format_base` module allows us to use all the formats in a consistent manner (avoiding special cases). E.g., if we want to read a file of an unknown format we can simply:

```python
from omsi.dataformat import *
formats = file_reader_base.file_reader_base.available_formats()
filename = 'my_unknown_file'
filereader = None
formatname = None
for fname, fclass in formats.items():
    if fclass.is_valid_dataset(filename):
        filereader = fclass
        formatname = fname
if filereader is not None:
    print "Using "+str(formatname)+" to read the file"
    openfile = filereader(basename=filename, requires_slicing=True)
```
CHAPTER
FOUR

DEVELOPING A NEW ANALYSIS FOR BASTET

BASTet provides developers with two easy ways to integrate new analyses. First, the simplest and most basic solution is to wrap analysis functions using either the @bastet_analysis Python decorator or by explicitly wrapping a function via wrapped funct = analysis generic.from function(my function). Second, to easily share and fully integrate an analysis with BASTet, we then create a new class that inherits from BASTet’s base analysis interface class.

4.1 Wrapping a function: The quick-and-dirty way

Sometimes developers just want to debug some analysis function or experiment with different variants of a code. At the same time, we want to be able to track the results of these kind of experiments in a simple fashion. The omsi.analysis.generic() provides us with such a quick-and-dirty solution. We say quick-and-dirty because it sacrifices some generality and features in favor for a very simple process.

Using the omsi.analysis.generic.analysis_generic.from_function() or omsi.analysis.generic.bastet_analysis() decorator, we can easily construct a generic omsi.analysis.base.analysis_base instance container object for a given function. We can then use this container object to execute our function, while tracking its provenance as well as save the results to file as we would with any other analysis object. This approach allows us to easily track, record, safe, share and reproduce code experiments with only minimal extra effort needed. Here we briefly outline the two main options to do this.

NOTE: Wrapping functions directly is not recommended for production workflows but is intended for development and debugging purposes only. This mechanism relies on that the library does the right thing in automatically determining input parameters, outputs, and their types and that we can handle all those types in the end-to-end process, from definition to storage. We do our best to make this mechanism work with a broad set of cases but we do not guarantee that the simple wrapping always work.

4.1.1 Option 1: Explicitly track specific executions of a function

Instead of calling our analysis function f() directly, we create an instance of omsi.analysis.generic.analysis_generic() via g = analysis_generic.from_function(f) which we then use instead of our function. To execute our function we can now either call g.execute(...) as usual or treat g as a function and call it directly g(...)

```python
import numpy as np
from omsi.analysis.generic import analysis_generic

# Define some example function we want to wrap to track results
def mysum(a):
    return np.sum(a)

# Create an analysis object for our function
```
4.1.2 Option 2: Implicitly track the last execution of a function

If we are only interested in recording the last execution of our function, then we can alternatively wrap our function directly using the @bastet_analysis decorator. The main difference between the two approaches is that using the decorator we only record the last execution of our function, while using the explicit approach of option 1, we can create as many wrapped instances of our functions as we want and track the execution of each independently.

```python
import numpy as np
from omsi.shared.log import log_helper
log_helper.set_log_level('DEBUG')
from omsi.analysis.generic import bastet_analysis

# Define some example function and wrap it
@bastet_analysis
def mysum(a):
    """Our own sum function""
    return np.sum(a)

# mysum['a'] = np.arange(10)
# As with the any wrapped function we can delay execution and set
# parameters before we actually run the analysis

# Execute the analysis
res = mysum(a=np.arange(10))  # Or we can set parameters and execute in the same call
```

See omsi.analysis.generic.bastet_analysis() for further details on additional optional inputs of the wrapping, e.g., to specify additional information about inputs and outputs.

4.1.3 Example 1: Defining and using wrapped functions

The code example shown below illustrates the “wrapping” of a simple example function mysum(a), which simply uses numpy.sum to compute the sum of objects in an array. (NOTE: We could naturally also use numpy.sum directly, we use mysum(a) mainly to illustrate that this approach also works with functions defined in the interpreter.)

```python
import numpy as np
from omsi.shared.log import log_helper
log_helper.set_log_level('INFO')
from omsi.analysis.generic import bastet_analysis
from omsi.dataformat.omsi_file.main_file import omsi_file

# Define some example function we want to wrap to track results
def mysum(a):
    return np.sum(a)

# Create an analysis object for our function
g = analysis_generic.from_function(mysum)

# Execute the analysis
```
res = g.execute(a=np.arange(10))
log_helper.log_var(__name__, res=res)  # Logging the result

# Save the analysis to file
f = omsi_file('autowrap_test.h5', 'a')
e = f.create_experiment()
exp_index = e.get_experiment_index()
ana_obj, ana_index = e.create_analysis(g)
# Close the file
f.flush()
del f

# Restore the analysis from file
f = omsi_file('autowrap_test.h5', 'a')
e = f.get_experiment(exp_index)
a2 = e.get_analysis(ana_index)
g2 = a.restore_analysis()
res2 = g2.execute()
log_helper.log_var(__name__, res2=res2)  # Logging the result
if res == res2:
    log_helper.info(__name__, "CONGRATULATIONS---The results matched")
else:
    log_helper.error(__name__, "SORRY---The results did not match")

When we run our script, we can see that we were able to successfully capture the execution of our function and recreate
the analysis from file.

```
machine:dir username$ python autowrap_function.py
2015-09-29 15:21:32,729 - __main__ - INFO - res = 45
2015-09-29 15:21:32,778 - __main__ - INFO - res2 = 45
2015-09-29 15:21:32,778 - __main__ - INFO - CONGRATULATIONS---The results matched
```

The figure below shows a view of the file generated by our wrapped function execution example shown above.

Note, we can use the wrapped function objects as usual in an analysis workflow to combine our functions with other
analyses. For example, the simple example shown below shows how we can quickly define a simple filter to set all
intensities that are less than 10 to a value of 0 before executing an analysis. We here first execute global peak finding
to reduce the data, than apply a simple wrapped filter function to filter the data values, and then compute NMF on the
filtered data.

```
from omsi.dataformat.omsi_file import *
from omsi.analysis.findpeaks.omsi_findpeaks_global import omsi_findpeaks_global
from omsi.analysis.multivariate_stats.omsi_nmf import omsi_nmf
from omsi.analysis.generic import analysis_generic
import numpy as np

f = omsi_file('/Users/oruebel/Devel/openmsi-data/msidata/20120711_Brain.h5', 'r')
d = f.get_experiment(0).get_msidata(0)

# Specify the analysis workflow
al = omsi_findpeaks_global()
al['msidata'] = d
al['mzdata'] = d.mz

# Wrap a simple function to filter all peaks with less than 10 counts
def f(a):
a[a<10] = 0
return a

a2 = analysis_generic.from_function(f)
```

4.1. Wrapping a function: The quick-and-dirty way
a2['a'] = a1['peak_cube']  # Use the peak_cube from a1 as input for the filter

# Create an NMF for the filtered data
a3 = omsi_nmf()
a3['msidata'] = a2['output_0']  # Make the output of our analysis the input of the NMF
a3['numIter'] = 2

# Run our simple workflow, i.e.,: peak_finder --> our_filter --> nmf
a3.execute_recursive()

By default, the outputs are named and numbered using the schema output_#, i.e., in the above example we used a2['output_0'] to access the output our wrapped function. To define user-defined names for the outputs of a wrapped function we can simply provide a list of strings to the input parameter output_names of the analysis_generic.from_function(...).

### 4.2 Writing a new analysis class for BASTet – Part 1: Overview

The OpenMSI Toolkit includes a basic template for implementing new analyses as part of OpenMSI. The template is located in omsi.templates.omsi_analysis_template.py. The template provides step-by-step instructions on how to implement a new analysis. Simply search top-to-bottom for EDIT_ME markers to find locations that need to be edited and what changes need to be made.

The implementation of a new analysis is divided into three main steps. We here provide a brief overview of these steps. A detailed walk-through the required implementation is provided in the following sections.

**Step 1) Basic integration of your analysis with OpenMSI (Required)**

The basic integration is simple and should requires only minimal additional effort. The basic integration with the OpenMSI provides:
• full integration of the analysis with the OpenMSI file format and API
• full support for OpenMSI's data provenance capabilities
• full integration with analysis drivers (e.g. the command line driver) enabling direct execution of the analysis with automatic handling of user input specification, help, etc.
• basic integration of the analysis with the website, in that a user will be able to browse the analysis in the online file browser. The basic integration automatically provides full support for the qmetadata and qcube URL data access patterns. The basic integration provides limited support for the qslice, qspectrum, and qmz patterns, in that it automatically exposes all dependencies of the analysis that support these patterns but it does not expose the data of the analysis itself. This is part of step 2.

Step 2) Integrating your analysis with the OpenMSI web-based viewer (Recommended)
Once the basic integration is complete, you may want integrate your analysis fully with the OpenMSI online viewer, in order to make your analysis easily accessible to the OpenMSI user community. This step requires the implementation of the qslice, qspectrum, and qmz URL patterns for the analysis. This step completes the integration with the OpenMSI framework itself.

Step 3) Making your analysis self-sufficient (Recommended)
This step makes your analysis “self-sufficient” in that it allows you to execute your analysis from the command-line. This step is usually very simple as we can just use BASTet’s integrated analysis driver to do the job for us.

4.2.1 Some important features of analysis_base

omsi.analysis.analysis_base is the base class for all omsi analysis functionality. The class provides a large set of functionality designed to facilitate i) storage of analysis data in the omsi HDF5 file format and ii) integration of new analysis capabilities with the OpenMSI web API and the OpenMSI web-based viewer (see Viewer functions below for details), iii) support for data provenance, and iv) in combination with the omsi_analysis_driver module enable the direct execution of analysis, e.g. from the command line.

Slicing

analysis_base implements basic slicing to access data stored in the main member variables. By default the data is retrieved from __data_list by the __getitem__(key) function, which implements the [..] operator, i.e., the functions returns __data_list[key]['data']. The key is a string indicating the name of the parameter to be retrieved. If the key is not found in the __data_list then the function will try to retrieve the data from __parameter_list instead. By adding “parameter/key” or “dependency/key” one may also explicitly retrieve values from the __parameter_list and __dependency_list.

Important Member Variables

• analysis_identifier defines the name for the analysis used as key in search operations.
• __data_list defines a list of omsi.analysis.analysis_data object objects to be written to the HDF5 file. Derived classes need to add all data that should be saved for the analysis in the omsi HDF5 file to this dictionary. See omsi.analysis.analysis_data for details.
• parameters List of parameter_data to be written to the HDF5 file. Derived classes need to add all parameter data that should be saved for the analysis in the omsi HDF5 file to this dictionary using the provided add_parameter(...) function. See omsi.analysis.analysis_data and add_parameter(...) function of analysis_base for details.
**I/O functions**

These functions can be optionally overwritten to control how the analysis data should be written/read from the omsi HDF5 file. Default implementations are provided here, which should be sufficient for most cases.

- **write_analysis_data**: By default all data is written by omsi.dataformat.omsi_file.analysis.omsi_file_analysis. By implementing this function we can implement the write for the main data (i.e., what is stored in self.__data_list) ourselves. In practice (at least in the serial case) this should not be needed. However, overwriting the function can be useful when implementing an analysis using MPI and we want to avoid gathering the data on rank the root rank (usually rank 0).

- **add_custom_data_to_omsi_file**: The default implementation is empty as the default data write is managed by the omsi_file_experiment.create_analysis() function. Overwrite this function, in case that the analysis needs to write data to the HDF5 omsi file beyond what the default omsi data API does.

- **read_from_omsi_file**: The default implementation tries to reconstruct the original data as far as possible, however, in particular in case that a custom add_custom_data_to_omsi_file function has been implemented, the default implementation may not be sufficient. The default implementation reconstructs: i) analysis_identifier and reads all custom data into ii)__data_list. Note, an error will be raised in case that the analysis type specified in the HDF5 file does not match the analysis type specified by get_analysis_type(). This function can be optionally overwritten to implement a custom data read.

**Web API Functions**

Several convenient functions are used to allow the OpenMSI online viewer to interact with the analysis and to visualize it. The default implementations provided here simply indicate that the analysis does not support the data access operations required by the online viewer. Overwrite these functions in the derived analysis classes in order to interface them with the viewer. All viewer-related functions start with `v_`.

NOTE: the default implementation of the viewer functions defined in `analysis_base` are designed to take care of the common requirement for providing viewer access to data from all dependencies of an analysis. In many cases, the default implementation is often still called at the end of custom viewer functions.

NOTE: The viewer functions typically support a `viewerOption` parameter. `viewerOption=0` is expected to refer to the analysis itself.

- **v_qslice**: Retrieve/compute data slices as requested via qslice URL requests. The corresponding view of the DJANGO data access server already translates all input parameters and takes care of generating images/plots if needed. This function is only responsible for retrieving the data.

- **v_qspectrum**: Retrieve/compute spectra as requested via qspectrum URL requests. The corresponding view of the DJANGO data access server already translates all input parameters and takes care of generating images/plots if needed. This function is only responsible for retrieving the data.

- **v_qmz**: Define the m/z axes for image slices and spectra as requested by qspectrum URL requests.

- **v_qspectrum_viewer_options**: Define a list of strings, describing the different viewer options available for the analysis for qspectrum requests (i.e., `v_qspectrum`). This feature allows the analysis developer to define multiple different visualization modes for the analysis. For example, when performing a data reduction (e.g., PCA or NMF) one may want to show the raw spectra or the loadings vector of the projection in the spectrum view (v_qspectrum). By providing different viewer options we allow the user to decide which option they are most interested in.

- **v_qslice_viewer_options**: Define a list of strings, describing the different viewer options available for the analysis for qslice requests (i.e., `v_qslice`). This feature allows the analysis developer to define multiple
different visualization modes for the analysis. For example, when performing a data reduction (e.g., PCA or NMF) one may want to show the raw spectra or the loadings vector of the projection in the spectrum view (\texttt{v_qspectrum}). By providing different viewer options we allow the user to decide which option they are most interested in.

**Executing, saving, and restoring an analysis object**

Using the command-line driver we can directly execute analysis as follows:

```
python omsi/analysis/omsi_analysis_driver <analysis_module_class> <analysis_parameters>
```

E.g. to execute a non-negative matrix factorization (NMF) using the `omsi.analysis.multivariate_stats.omsi_nmf` module we can simply:

```
python omsi/analysis/omsi_analysis_driver.py multivariate_stats/omsi_nmf.py
--msidata "test_brain_convert.h5:/entry_0/data_0"
--save "test_ana_save.h5"
```

Any analysis based on the infrastructure provided by `analysis_base` is fully integrated with OpenMSI file API provided by `omsi.dataformat.omsi_file`. This means the analysis can be directly saved to an OMSI HDF5 file and the saved analysis can be restored from file. In OMSI files, analyses are generally associated with experiments, so that we use the `omsi.dataformat.omsi_file.omsi_file_experiment` API here.

```python
# Open the MSI file and get the desired experiment
from omsi.dataformat.omsi_file import *
f = omsi_file( filename, 'a' )
e = f.get_experiment(0)

# Execute the analysis
d = e.get_msidata(0)
a = omsi_myanalysis()
a.execute(msidata=d, integration_width=10, msidata_dependency=d)

# Save the analysis object.
analysis_object , analysis_index = exp.create_analysis( a )

# This single line is sufficient to store the complete analysis to the omsi file.
# By default the call will block until the write is complete. Setting the
# parameter flushIO=False enables buffered write, so that the call will
# return once all data write operations have been scheduled. Here we get
# an omsi.dataformat.omsi_file.omsi_file_analysis
# object for management of the data stored in HDF5 and the integer index of the analysis.

# To restore an analysis from file, i.e., read all the analysis data from file
# and store it in a corresponding analysis object we can do the following.
# Similar to the read_from_omsi_file(...) function of analysis_base
# mentioned below, we can decide via parameter settings of the function,
# which portions of the analysis should be loaded into memory
a2 = analysis_object.restore_analysis()

# If we want to now re-execute the same analysis we can simply call
a2.execute()

# If we want to rerun the analysis but change one or more parameter settings,
# then we can simply change those parameters when calling the execute function
d2 = e.get_msidata(1)  # Get another MSI dataset
a2.execute(msidata=d2)  # Execute the analysis on the new MSI dataset
```

4.2. Writing a new analysis class for BASTet – Part 1: Overview
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

# The omsi_file_analysis class also provides a convenient function that allows us
to recreate, i.e., restore and run the analysis, in a single function call
a3 = analysis_object.recreate_analysis()

# The recreate_analysis(...) function supports additional keyword arguments
# which will be passed to the execute(...) call of the analysis, so that we
can change parameter settings for the analysis also when using the
# recreate analysis call.

# If we know the type of analysis object (which we can also get from file), then we
# naturally also restore the analysis from file ourselves via
a4 = omsi_myanalysis().read_from_omsi_file(analysisGroup=analysis_object,
   load_data=True, 
   load_parameters=True,
   load_runtime_data=True, 
   dependencies_omsi_format=True )

# By setting load_data and/or load_parameters to False, we create h5py instead of
# numpy objects, avoiding the actual load of the data. CAUTION: To avoid the accidental
# overwrite of data we recommend to use load_data and load_parameters as False only
# when the file has been opened in read-only mode ’r’.

# Rerunning the same analysis again
a4.execute()

### 4.3 Writing a new analysis class for BASTet – Part 2: Analysis Template

In this section we describe the basic development of a new analysis using the analysis template provided by BASTet.

#### 4.3.1 Step 1) Basic integration

The simple steps outlined below provide you now with full integration of your analysis with the OpenMSI file format and API and full support for OpenMSI’s data provenance capabilities. It also provides basic integration of your analysis with the OpenMSI website, in that a user will be able to browse your analysis in the online file browser. The basic integration also automatically provides full support for the qmetadata and qcube URL data access patterns, so that you can start to program against your analysis remotely. The basic integration provides limited support for the qslice, qspectrum, and qmz patterns, in that it automatically exposes all dependencies of the analysis that support these patterns but it does not expose the data of your analysis itself. This is part of step 2. Once you have completed the basic integration your final analysis code should look something like this:

```python
class omsi_mypeakfinder(analysis_base) :
    def __init__(self, name_key="undefined"):
        """Initialize the basic data members""
        super(omsi_mypeakfinder, self).__init__()

        self.analysis_identifier = name_key
        # Define the names of the outputs generated by the analysis
        self.data_names = [ 'peak_cube' , 'peak_mz' ]

        # Define the input parameters of the analysis
dtypes = self.get_default_dtypes()
```

Chapter 4. Developing a new Analysis for BASTet
groups = self.get_default_parameter_groups()
self.add_parameter(name='msidata',
    help='The MSI dataset to be analyzed',
    dtype=dtypes['ndarray'],
    group=groups['input'],
    required=True)
self.add_parameter(name='mzdata',
    help='The m/z values for the spectra of the MSI dataset',
    dtype=dtypes['ndarray'],
    group=groups['input'],
    required=True)
self.add_parameter(name='integration_width',
    help='The window over which peaks should be integrated',
    dtype=float,
    default=0.1,
    group=groups['settings'],
    required=True)
self.add_parameter(name='peakheight',
    help='Peak height parameter',
    dtype=int,
    default=2,
    group=groups['settings'],
    required=True)

def execute_analysis(self):
    """..."""
    # Copy parameters to local variables. This is purely for convenience and is not mandatory.
    # NOTE: Input parameters are automatically record (i.e., we don't need to do anything special.
    msidata = self['msidata']
mzdata = self['mzdata']
inintegration_width = self['integration_width']
peakheight = self['peakheight']

    # Implementation of my peakfinding algorithm
    ...

    # Return the result.
    # NOTE: We need to return the output in the order we specified them in self.data_names
    # NOTE: The outputs will be automatically recorded (i.e., we don't need to do anything special)
    return peakCube, peakMz
self['peak_cube'] = peakCube

    ...

# Defining a main function is optional. However, allowing a user to directly execute your analysis
# from the command line is simple, as we can easily reuse the command-line driver module
if __name__ == "__main__":
    from omsi.analysis.omsi_analysis_driver import cl_analysis_driver
    cl_analysis_driver(analysis_class=omsi_mypeakfinder).main()

1.1 Creating a new analysis skeleton

- Copy the analysis template to the appropriate location where your analysis should live (NOTE: The analysis
template may have been updated since this documentation was written). Any new analysis should be located
in a submodule of the omsi.analysis. module. E.g., if you implement a new peak finding algorithm, it
should be placed in omsi/analysis/findpeaks. For example:

```bash
cp omsi/templates/omsi_analysis_template.py openmsi-tk/omsi/analysis/findpeaks/omsi_mypeakfinder.py
```

- Replace all occurrences of `omsi_analysis_template` in the file with the name of your analysis class, e.g., `omsi_mypeakfinder`. You can do this easily using “Replace All” feature of most text editors. or on most Unix systems (e.g, Linux or MacOS) on the commandline via:

```bash
cd openmsi-tk/omsi/analysis/findpeaks
sed -i.bak 's/omsi_analysis_template/omsi_mypeakfinder/' omsi_mypeakfinder.py
rm omsi_mypeakfinder.py.bak
```

- Add your analysis to the `__init__.py` file of the python module where your analysis lives. In the `__init__.py` file you need to add the name of your analysis class to the `all__` list and add a import of your class, e.g, `from omsi_mypeakfinder import *`. For example:

```python
all__ = [ "omsi_mypeakfinder", "omsi_findpeaks_global", ...]
from omsi_findpeaks_global import *
from omsi_findpeaks_local import *
...
```

- The analysis template contains documentation on how to implement a new analysis. Simply search for `EDIT_ME` to locate where you should add code and descriptions of what code to add.

### 1.2 Specifying analysis inputs and outputs

In the `__init__` function specify the names of the input parameters of your analysis as well as the names of the output data generated by your analysis. Note, the `__init__` function should have a signature that allows us to instantiate the analysis without having to provide any inputs. E.g.,

```python
def __init__(self, name_key="undefined"):
    """Initialize the basic data members""
    super(omsi_mypeakfinder, self).__init__()
    self.analysis_identifier = name_key

    # Define the names of the outputs
    self.data_names = ['peak_cube', 'peak_mz']

    # Define the input parameters
    dtypes = self.get_default_dtypes() # List of default data types. Build-in types are
    # available as well but can be safely used directly as well
    groups = self.get_default_parameter_groups() # List of default groups to organize parameters. We
    # to use the 'input' group for all input data to be
    # as this will make the integration with OpenMSI ea

    self.add_parameter(name='msidata',
                       help='The MSI dataset to be analyzed',
                       dtype=dtypes['ndarray'],
                       group=groups['input'],
                       required=True)
    self.add_parameter(name='mzdata',
                       help='The m/z values for the spectra of the MSI dataset',
                       dtype=dtypes['ndarray'],
                       group=groups['input'],
                       required=True)
    self.add_parameter(name='integration_width',
                       help='The window over which peaks should be integrated',
                       dtype=float,
                        ...)
1.3: Implementing the execute_analysis function

1.3.1 Document your execute_analysis function. OpenMSI typically uses Sphynx notation in the doc-string. The doc-string of the execute_analysis(...) function and the class are used by the analysis driver modules to provide a description of your analysis as part of the help and will also be included in the help string generated by the get_help_string() inherited via analysis_base function.

```python
def execute_analysis(self):
    """This analysis computes global peaks in MSI data..."
```

1.3.2 Implement your analysis. For convenience it is often useful to assign the your parameters to local variables, although, this is by no means required. Note, all values are stored as 1D+ numpy arrays, however, are automatically converted for you, so that we can just do:

```python
integration_width = self['integration_width']
```

1.3.4 Return the outputs of your analysis in the same order as specified in the self.data_names you specified in your __init__ function (here ['peak_cube', 'peak_mz']):

```python
return peakCube, peakMZ
```

Results returned by your analysis will be automatically saved to the respective output variables. This allows users to conveniently access your results and it enables the OpenMSI file API to save your results to file. We here automatically convert single scalars to 1D numpy arrays to ensure consistency. Although, the data write function can handle a large range of python built_in types by automatically converting them to numpy for storage in HDF5, we generally recommend to convert use numpy directly here to save your data.

With this you have now completed the basic integration of your analysis with the OpenMSI framework.

4.3.2 Step 2) Integrating the Analysis with the OpenMSI Web API:

Once the analysis is stored in the OMSI file format, integration with qmetadata and qcube calls of the web API is automatic. The qmetadata and qcube functions provide general purpose access to the data so that we can immediately start to program against our analysis.

Some applications—such as the OpenMSI web-based viewer—utilize the simplified, special data access patterns qslice, qsspectrum, and qmz in order to interact with the data. The default implementation of these function available in omsi.analysis.analysis_base exposes the data from all dependencies of the analysis that support these patterns. For full integration with the web API, however, we need to implement this functionality in our analysis class. The qmz pattern in particular is relevant to both the qslice and qsspectrum pattern and should be always implemented as soon as one of the two patterns is defined.
2.1 Implementing the qslice pattern

class omsi_myanalysis(analysis_base) :
...

@classmethod
def v_qslice(cls, anaObj, z, viewer_option=0):
    """Implement support for qslice URL requests for the viewer"""
    anaObj: The omsi_file_analysis object for which slicing should be performed.
z: Selection string indicating which z values should be selected.
viewer_option: An analysis can provide different default viewer behaviors
    for how slice operation should be performed on the data.
    This is a simple integer indicating which option is used.

    :returns: numpy array with the data to be displayed in the image slice
    viewer. Slicing will be performed typically like [:,:,zmin:zmax].
    """
    from omsi.shared.omsi_data_selection import *
    #Implement custom analysis viewer options
    if viewer_option == 0 :
        dataset = anaObj[ 'labels' ] #We assume labels was a 3D image cube of labels
        zselect = selection_string_to_object(z)
        return dataset[ :, :, zselect ]
    #Expose recursively the slice options for any data dependencies. This is useful
    #to allow one to trace back data and generate complex visualizations involving
    #multiple different data sources that have some from of dependency in that they
    #led to the generation of this analysis. This behavior is already provided by
    #the default implementation of this function in analysis_base.
    elif viewer_option >= 0 :
        #Note, the base class does not know out viewer_options so we need to adjust
        #the viewerOption accordingly by subtracting the number of our custom options.
        return super(omsi_myanalysis,cls).v_qslice( anaObj, z, viewer_option-1)
    else :
        return None

@classmethod
def v_qslice_viewer_options(cls, anaObj) :
    """Define which viewer_options are supported for qspectrum URL's"""
    #Get the options for all data dependencies
    dependent_options = super(omsi_findpeaks_global,cls).v_qslice_viewer_options(anaObj)
    #Define our custom viewer options
    re = ["Labels"] + dependent_options
    return re

NOTE: We here convert the selection string to a python selection (i.e., a list, slice, or integer) object using the
omsis.shared.omsi_data_selection.check_selection_string(...). This has the advantage
that we can use the given selection directly in our code and avoids the use of a potentially dangerous eval, e.g.,
return eval("dataset[:,:, %s]") % (z,)). While we can also check the validity of the selection string
using omsi.shared.omsi_data_selection.check_selection_string(...), it is recommended to
convert the string to a valid python selection to avoid possible attacks.
2.2 Implementing the qspectrum pattern

```python
class omsi_myanalysis(analysis_base):
    ...
    @classmethod
    def v_qspectrum(cls, anaObj, x, y, viewer_option=0):
        r"""Implement support for qspectrum URL requests for the viewer.
anaObj: The omsi_file_analysis object for which slicing should be performed.
x: x selection string
y: y selection string
viewer_option: If multiple default viewer behaviors are available for a given
analysis then this option is used to switch between them.

:returns: The following two elements are expected to be returned by this function:
1) 1D, 2D or 3D numpy array of the requested spectra. NOTE: The spectrum axis,
e.g., mass (m/z), must be the last axis. For index selection x=1,y=1 a 1D array
is usually expected. For indexList selections x=[0] & y=[1] usually a 2D array
is expected. For range selections x=0:1 & y=1:2 we one usually expect a 3D array.
This behavior is consistent with numpy and h5py.
2) None in case that the spectra axis returned by v_qmz are valid for the
returned spectrum. Otherwise, return a 1D numpy array with the m/z values
for the spectrum (i.e., if custom m/z values are needed for interpretation
of the returned spectrum). This may be needed, e.g., in cases where a
per-spectrum peak analysis is performed and the peaks for each spectrum
appear at different m/z values.

Developer Note: h5py currently supports only a single index list. If the user provides
an index-list for both x and y, then we need to construct the proper merged list and
load the data manually, or, if the data is small enough, one can load the full data
into a numpy array which supports multiple lists in the selection. This, however, is
only recommended for small datasets.
""

    data = None
    customMZ = None
    if viewer_option == 0:
        from omsi.shared.omsi_data_selection import *
        dataset = anaObj['labels']
        if (check_selection_string(x) == selection_type['indexlist']) and 
           (check_selection_string(y) == selection_type['indexlist']):
            # Assuming that the data is small enough, we can handle the multiple list
            # selection case here just by loading the full data use numpy to do the
            # subselection. Note, this version would work for all selection types but
            # we would like to avoid loading the full data if we don't have to.
            xselect = selection_string_to_object(x)
            yselect = selection_string_to_object(y)
            data = dataset[:,xselect,yselect,:]
            # Since we already confirmed that both selection strings are index lists we could
            # also just do an eval as follows.
            # data = eval("dataset[:,%s,%s,:]" % (x,y))
        else:
            xselect = selection_string_to_object(x)
            yselect = selection_string_to_object(y)
            data = dataset[xselect,yselect,:]
```

4.3. Writing a new analysis class for BASTet – Part 2: Analysis Template 25
2.3 Implementing the *qmz* pattern

```python
class omsi_myanalysis(analysis_base):
    ...

@classmethod
def v_qmz(cls, anaObj, qslice_viewer_option=0, qspectrum_viewer_option=0):
    """Implement support for qmz URL requests for the viewer."

    Get the mz axes for the analysis

    anaObj: The omsi_file_analysis object for which slicing should be performed.
    qslice_viewer_option: If multiple default viewer behaviors are available for
                        a given analysis then this option is used to switch between them
                        for the qslice URL pattern.
    qspectrum_viewer_option: If multiple default viewer behaviors are available
                            for a given analysis then this option is used to switch between
                            them for the qspectrum URL pattern.

    :returns: The following four arrays are returned by the analysis:
    - mzSpectra : 1D numpy array with the static mz values for the spectra.
    - labelSpectra : String with label for the spectral mz axis
    - mzSlice : 1D numpy array of the static mz values for the slices or
                None if identical to the mzSpectra array.
    - labelSlice : String with label for the slice mz axis or None if identical to labelSpectra.
    - valuesX: The values for the x axis of the image (or None)
    - labelX: Label for the x axis of the image
    - valuesY: The values for the y axis of the image (or None)
    - labelY: Label for the y axis of the image
    - valuesZ: The values for the z axis of the image (or None)
    - labelZ: Label for the z axis of the image
```
Developer Note: Here we need to handle the different possible combinations for the different viewer_option patterns. It is in general safe to populate mzSlice and labelSlice also if they are identical with the spectrum settings, however, this potentially has a significant overhead when the data is transferred via a slow network connection, this is why we allow those values to be None in case that they are identical.

###

# The four values to be returned
mzSpectra = None
labelSpectra = None
mzSlice = None
labelSlice = None

peak_cube_shape = anaObj['labels'].shape  # We assume labels was a 3D image cube of labels
valuesX = range(0, peak_cube_shape[0])
labelX = 'pixel index X'
valuesY = range(0, peak_cube_shape[1])
labelY = 'pixel index Y'
valuesZ = range(0, peak_cube_shape[2]) if len(peak_cube_shape) > 3 else None
labelZ = 'pixel index Z' if len(peak_cube_shape) > 3 else None

# Both qslice and qspectrum here point to our custom analysis
if qspectrumViewerOption == 0 and qsliceViewerOption==0:
    # Loadings
    mzSpectra = anaObj['labels'][:, :]
    labelSpectra = "Labels"

# Both viewer options point to a data dependency
elif qspectrumViewerOption > 0 and qsliceViewerOption>0:
    mzSpectra, labelSpectra, mzSlice, labelSlice = \
    super(omsi_findpeaks_global,cls).v_qmz( anaObj, \
    qsliceViewerOption-1, qspectrumViewerOption-1)

# Only the a qslice options point to a data dependency
elif qspectrumViewerOption == 0 and qsliceViewerOption>0:
    mzSpectra = anaObj['peak_mz'][:, :]
    labelSpectra = "m/z"
    tempA, tempB, mzSlice, labelSlice, valuesX, labelX, valuesY, labelY, valuesZ, labelZ = \
    super(omsi_findpeaks_global,cls).v_qmz( anaObj, \
    qsliceViewerOption-1, 0)

# Only the qspectrum option points to a data dependency
elif qspectrumViewerOption > 0 and qsliceViewerOption==0:
    mzSlice = anaObj['peak_mz'][:, :]
    labelSlice = "m/z"
    # Ignore the spatial axes and slice axis as we use our own
    super(omsi_findpeaks_global,cls).v_qmz( anaObj, \
    0, qspectrumViewerOption-1)

return mzSpectra, labelSpectra, mzSlice, labelSlice, valuesX, labelX, valuesY, labelY, valuesZ, labelZ

###

#### 4.3.3 Step 3) Making your analysis self-sufficient

Making your analysis self-sufficient is trivial. If you used the analysis template provided by the toolkit, then you have already completed this step for free. In order to allow a user to run our analysis from the command line we need a main function. We here can simply reuse the command line driver provided by the toolkit. Using the command line driver we can run the analysis via:
python omsi/analysis/omsi_analysis_driver.py findpeaks.omsi_mypeakfinder
   --msidata "test_brain_convert.h5:/entry_0/data_0"
   --mzdata "test_brain_convert.h5:/entry_0/data_0/mz"
   --save "test_ana_save.h5"

To now enable us to execute our analysis module itself we simply need to add the following code (which is already part of the template)

```python
if __name__ == "__main__":
    from omsi.analysis.omsi_analysis_driver import cl_analysis_driver
    cl_analysis_driver(analysis_class=omsi_mypeakfinder).main()
```

With this we can now directly execute our analysis from the command line, get a command-line help, specify all our input parameters on the command line, and save our analysis to file. To run the analysis we can now do:

```plaintext
python omsi/analysis/findpeaks/omsi_findpeaks_global.py
   --msidata "test_brain_convert.h5:/entry_0/data_0"
   --mzdata "test_brain_convert.h5:/entry_0/data_0/mz"
   --save "test_ana_save.h5"
```

This will run our peak finder on the given input data and save the result to the first experiment in the test_ana_save.h5 (the output file will be automatically created if it does not exist).

The command line driver also provides us a well-formated help based on the our parameter specification and the doc-string of the analysis class and its execute_analysis(...) function. E.g:

```python
>>> python omsi/analysis/findpeaks/omsi_findpeaks_global.py --help
usage: omsi_findpeaks_global.py [-h] [--save SAVE] --msidata MSIDATA --mzdata MZDATA [--integration_width INTEGRATION_WIDTH]
                                  [--peakheight PEAKHEIGHT]
                                  [--slwindow SLWINDOW]
                                  [--smoothwidth SMOOTHWIDTH]

class description:
Basic global peak detection analysis. The default implementation computes the peaks on the average spectrum and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.

TODO: The current version assumes 2D data

execution description:
Execute the global peak finding for the given msidata and mzdata.

optional arguments:
  -h, --help           show this help message and exit
  --save SAVE          Define the file and experiment where the analysis should be stored. A new file will be created if the given file does not exists but the directory does. The filename is expected to be of the from: <filename>:<entry_#> . If no experiment index is given, then experiment index 0 (i.e, entry_0) will be assumed by default. A validpath may, e.g, be "test.h5:/entry_0" or jus "test.h5" (default: None)
```
4.4 Writing a new analysis class for BASTet – Part 3: Customizing core features

4.4.1 Custom data save

In most cases the default data save and restore functions should be sufficient. However, the `analysis_base` API also supports implementation of custom HDF5 write. To extend the existing data write code, simple implement the following function provided by `analysis_base`.

```python
def add_custom_data_to_omsi_file(self, analysisGroup):
    # This function can be optionally overwritten to implement a custom data write
    # function for the analysis to be used by the omsi_file API.
    
    Note, this function should be used only to add additional data to the analysis
    group. The data that is written by default is typically still written by the
    omsi_file_experiment.create_analysis() function, i.e., the following data is
    written by default: i) analysis_identifier, ii) get_analysis_type,
    iii) __data_list, iv) __parameter_list, v) __dependency_list. Since the
    omsi_file_experiment.create_analysis() functions takes care of setting up the
    basic structure of the analysis storage (including the subgroups for storing
    parameters and data dependencies) this setup can generally be assumed to exist
    before this function is called. This function is called automatically at the
    end omsi_file_experiment.create_analysis() (i.e., actually
```

This command-line tool has been auto-generated using the OpenMSI Toolkit.
4.4.2 Custom analysis restore

Similarly in order implement custom data restore behavior we can overwrite the default implementation of `omsi.analysis.analysis_base.analysis_base.read_from_omsi_file()` . In this case one will usually call the default implementation via `super(omsi_myanalysis,self).read_from_omsi_file(...)` first and then add any additional behavior.

4.4.3 Custom analysis execution

Analysis are typically executed using the `omsi.analysis.analysis_base.analysis_base.execute()` function we inherit from `py:class:omsi.analysis.analysis_base.analysis_base`. The `execute()` function controls many pieces, from recording and defining input parameters and outputs to executing the actual analysis. We, therefore, for NOT recommend to overwrite the `execute()` function, but rather to customize specific portions of the execution. To do this, `execute()` is broken into a number of functions which are called in a specific order. In this way we can easily overwrite select functions to customize a particular feature without having to overwrite the complete `execute()` function.

**Customizing setting of parameters**

First, the `execute` function uses `omsi.analysis.analysis_base.analysis_base.update_analysis_parameters()` to set all parameters that have been passed to execute accordingly. The default implementation of `update_analysis_parameters()`, hence, simply calls `self.set_parameter_values(...)` to set all parameter values. We can customize this behavior simply by overwriting the `update_analysis_parameters(...)` function.

**Customizing setting of default settings**

Second, the `execute` function uses the `omsi.analysis.analysis_base.analysis_base.define_missing_parameters()` function to set any required parameters that have not been set by the user to their respective values. Overwrite this function to customize how default parameter values are determined/set.

**Customizing the recording of runtime information**

The recording of runtime information is performed using the `omsi.shared.run_info_data.run_info_dict()` data structure. This data structure provides a series of functions that are called in order, in particular:

- `omsi.shared.run_info_data.run_info_dict.clear()`: This function is called first to clear the runtime dictionary. This is the same as the standard `dict.clear`.
• `osmi.shared.run_info_data.run_info_dict.record_preexecute()` : This function is called before the `execute_analysis` function is called and records basic system information.

• `osmi.shared.run_info_data.run_info_dict.record_postexecute()` : This function is called after the `execute_analysis` function has completed to record additional information, e.g., the time and duration of the analysis.

• `osmi.shared.run_info_data.run_info_dict.runinfo_clean_up()` : This function is called at the end to clean up the recorded runtime information. By default, `runinfo_clean_up()` removes any empty entries, i.e., key/value pairs where the value is either None or an empty string.

We can customize any of these functions by implementing a derived class of `osmi.shared.run_info_data.run_info_dict` where we can overwrite the functions. In order to use our derived class we can then assign our object to `osmi.analysis.analysis_base.analysis_base.run_info()`. This design allows us to modularly use the runtime information tracking also for other tasks, not just with our analysis base infrastructure.

**Customizing the analysis execution**

The analysis is completely implemented in the `osmi.analysis.analysis_base.analysis_base.execute_analysis()` function, which we have to implement in our derived class, i.e., running the analysis is fully custom anyways.

**Customizing the recording of analysis outputs**

Finally (i.e., right before returning analysis results), `execute(..)` uses the `osmi.analysis.analysis_base.analysis_base.record_execute_analysis_outputs()` function to save all analysis outputs. Analysis outputs are stored in the `self.__data_list` variable. We can save analysis outputs simply by slicing and assignment, e.g., `self[output_name] = my_output`. By overwriting `record_execute_analysis_outputs(...)` we can customize the recording of data outputs.
Figure *Illustration of an example workflow for image normalization*, illustrates the basic steps of using analysis workflows, i.e.,:

1. Create the analysis tasks
2. Define the analysis inputs
3. Execute

In the following we will use a simple analysis—workflow in which we compute a peak-cube from a raw MSI dataset and then compute an NMF from the peak cube—to illustrate the main steps involved for performing complex analysis workflows.
5.1 Step 1: Create the analysis tasks:

First we need to create our main analysis objects.

```python
from omsi.dataformat.omsi_file import *
from omsi.analysis.findpeaks.omsi_findpeaks_global import omsi_findpeaks_global
from omsi.analysis.multivariate_stats.omsi_nmf import omsi_nmf

# Open a file to get some MSI data
f = omsi_file('/Users/oruebel/Devel/openmsi-data/msidata/20120711_Brain.h5', 'r')
d = f.get_experiment(0).get_msidata(0)

# Specify the analysis workflow
# Create a global peak finding analysis
a1 = omsi_findpeaks_global() # Create the analysis
# Create an NMF that processes our peak cube
a2['numIter'] = 2            # Set input to perform 2 iterations only
```

5.2 Step 2: Define analysis inputs:

We can define the input parameters of analysis simply using standard dict-like assignment. Any dependencies between analysis tasks or OpenMSI files are created automatically for us.

```python
# Define the inputs of the global peak finder
a1['msidata'] = d           # Set the input msidata
a1['mzdata'] = d.mz         # Set the input mz data
# Define the inputs of the NMF
a2['msidata'] = a1['peak_cube']  # Set the input data to the peak cube
a2['numIter'] = 2           # Set input to perform 2 iterations only
```

NOTE: So far we have only specified our workflow. We have not executed any analysis yet, nor have we loaded any actual data yet.

5.3 Step 3: Execute

Finally we need to execute our analyses. For this we have various options, depending on which parts of our workflow we want to execute.

5.3.1 Executing a single analysis

To execute a single analysis, we can simply call the `execute()` function of our analysis. Note, the execute may raise an `AnalysisReadyError` in case that the inputs of the analysis are not ready. E.g.:

```python
a2.execute()    # Will fail with `''AnalysisReadyError'`
```

```python
al.execute()    # Will successfully execute al
```

5.3.2 Executing a single sub-workflow

To execute a single analysis including any missing dependencies, we can simply call the `execute_recursive()` function. E.g.
a2.execute_recursive()  # Will successfully execute a1

The above will execute a1 as well as a2 since a2 depends on a1.

**NOTE:** Recursive execution will only execute other analyses that are actually needed to complete our analysis and analysis results of dependent analyses that have been executed before will be reused. E.g., if we would call a2.execute_recursive() again, then only a2 would be executed again.

**NOTE:** When executing multiple dependent analyses, then the execution is typically controlled by a workflow executor py:meth:`omsii.workflow.executor`. By default, execute_recursive() will automatically create a default driver. If we want to customize the driver to be used then we can simply assign a driver to the analysis before-hand by setting the py:var:`omsii_analysis_base_analysis_base_driver` instance variable.

### 5.3.3 Executing all analyses

To run all analyses that have been created—independent of whether they depend on each other or not—we can simply call `omsii_analysis_base_analysis_base_execute_all()`.

```python
a1.execute_all()  # Execute all analyses
```

The above will execute any analysis that have not up-to-date. **NOTE:** In contrast to py:meth:`omsii_analysis_base_analysis_base_execute` and py:meth:`omsii_analysis_base_analysis_base_execute_recursive`, this is a class-level method and not an object-method. Again, the function uses a workflow driver, which we can customise by providing as driver as input to the function.

### 5.3.4 Executing multiple sub-workflows

To explicitly execute a subset of analyses (and all their dependencies) we can explicitly define a driver for the workflow we want to execute:

```python
from omsii_workflow_driver_greedy_executor import greedy_executor
driver = greedy_executor()  # Create a driver
driver.add_analysis(a1)  # Add one or more analyses
driver.add_analysis(a2)
driver.execute()  # Execute the workflow and its dependencies
```

```python
driver2 = greedy_executor()
driver2.add_analysis_all()  # Add all analyses
driver2.execute()  # Execute all analyses
```

### 5.4 Example: Normalizing an image

The goal of this example is to 1) illustrate the general concepts of how we can define analysis workflows and 2) illustrate the use of simple wrapped functions in combination with integrated analytics to create complex analysis workflows. The example shown below defines a basic image normalization workflow in which we:

1. Compute a reduced peak cube from an MSI image using the global peak finding analysis provided by BASTet.
2. Use a simple wrapped function to compute the total intensity image for the peak cube dataset computed in step 1.
3. Use a simple wrapped function to normalize the peak cube computed in step 1 using the total intensity image computed in step 2.
Illustration of the basic image normalization workflow defined below:

```
import numpy as np
from omso.shared.log import log_helper
log_helper.set_log_level('DEBUG')
from omso.analysis.findpeaks.omso_findpeaks_global import omso_findpeaks_global
from omso.dataformat.omso_file.main_file import omso_file
from omso.analysis.generic import analysis_generic

# Define a simple function to compute the total intensity image
def total_intensity(msidata, axis=2):
    import numpy as np
    return np.sum(msidata, axis=axis)

# Define a simple function to normalize an MSI data cube by per-spectrum normalization factors
def normalize_intensities(msidata, normfactors):
    import numpy as np
    return msidata / normfactors[:,:,:,np.newaxis]

# Get an example MSI image
f = omso_file('/Users/oruebel/Devel/openmsi-data/msidata/20120711_Brain.h5', 'r')
d = f.get_experiment(0).get_msidata(0)

# Define the global peak finder
a1 = omso_findpeaks_global()
a1['msidata'] = d
a1['mzdata'] = d.mz

# Define compute of total intensity image
a2 = analysis_generic.from_function(analysis_function=total_intensity, output_names=['total_intensities'])
a2['msidata'] = a1['peak_cube']

# Define the normalization of the peak cube
a3 = analysis_generic.from_function(normalize_intensities)
a3['msidata'] = a1['peak_cube']
a3['normfactors'] = a2['total_intensities']

# To run the workflow we now have several basic options
#
# 1) a3.execute_recursive() : Recursively execute the last analysis and all its dependencies (i.e., a1, a2, a3)
# 2) a1.execute_all() : Tell any analysis to execute all available analyses (i.e., a1, a2, a3)
# 3) Create our own workflow driver to control the execution of the analyses
# 4) Manually call execute on a1, a2, and a3 in order of their dependencies
#
# Execute the workflow
```

Chapter 5. Defining and Executing Analysis Workflows
a3.execute_recursive()
Similar to the `omsi.workflow.driver.cl_analysis_driver` (and the corresponding tool `omsi.tools.run_analysis`) for running single analysis tasks, BASTet provides basic tools for executing complete workflows via the concept of workflow drivers. Users may implement their own drivers using the appropriate base classes `omsi.workflow.driver.base`.

Some basic drivers and tools are already available with BASTet, e.g., the `omsi.workflow.driver.cl_workflow_driver` module (and the corresponding tool `omsi.tools.run_workflow`) defines a driver for driving and executing one or multiple workflows defined via workflow scripts, directly from the command-line.

### 6.1 Workflow Scripts

Workflow scripts are regular python scripts that include the i) creation of the analysis objects, and ii) full or partial definition of analysis parameters but usually **NOT** the actual execution of any of the analyses. Following our example from earlier, we may simply save the following code in python source file, e.g, `normalize_image.py`.

```python
import numpy as np
from omsi.analysis.findpeaks.omsi_findpeaks_global import omsi_findpeaks_global
from omsi.dataformat.omsi_file.main_file import omsi_file
from omsi.analysis.generic import analysis_generic

# Define a simple function to compute the total intensity image
def total_intensity(msidata, axis=2):
    import numpy as np
    return np.sum(msidata, axis=axis)

# Define a simple function to normalize an MSI data cube by per-spectrum normalization factors
def normalize_intensities(msidata, normfactors):
    import numpy as np
    return msidata / normfactors[:,:,np.newaxis]

# Define the global peak finder
a1 = omsi_findpeaks_global()

# Define compute of total intensity image
a2 = analysis_generic.from_function(analysis_function=total_intensity,
                                      output_names=['total_intensities'])
a2['msidata'] = a1['peak_cube']

# Define the normalization of the peak cube
a3 = analysis_generic.from_function(normalize_intensities)
```
When using our command-line tool, all parameters that are not defined for any of the analyses are automatically exposed via command-line options. In contrast to our previous example, we here, e.g., do not set the input msidata and mzdata parameters for our global peak finder (a1). In this way, we can now easily set the input file we want to process directly via the command line. In cases where we want to expose a parameter via the command line but still want to provide a good default setting for the user, we can set the default value of a parameter via, e.g.,

```
a1.get_parameter_data_by_name('peakheight')['default'] = 3
```

To execute our above example from the command line we can now simply do the following:

```
python run_workflow.py --script normalize_image.py
  --ana_0:msidata $HOME/20120711_Brain.h5:/entry_0/data_0
  --ana_0:mzdata $HOME/20120711_Brain.h5:/entry_0/data_0/mz
```

In order to avoid collisions between parameters with the same name for different analyses, the tool prepends the unique analysis_identifier to each parameter. Since we did not set any explicit analysis_identifier (e.g., via `a1.analysis_identifier='a1'`), the tool automatically generated unique identifiers (i.e., ana_0, ana_1, and ana_3 for our 3 analyses). To view all available command line option we can simply call the script with --help. If one or more workflow scripts are given (here via separate --script parameters), then all unfilled options of those workflows and the corresponding analyses will be listed as. E.g.

```
newlappy:tools oruebel$ python run_workflow.py --script normalize_image.py --help
usage: run_workflow.py --script SCRIPT [--save SAVE] [--profile]
  [-h]

Execute analysis workflow(s) based on a given set of scripts

required arguments:
  --script SCRIPT  The workflow script to be executed. Multiple scripts
                  may be added via separate --script arguments (default: None)

optional arguments:
  --save SAVE      Define the file and experiment where all analysis
                  results should be stored. A new file will be created
                  if the given file does not exists but the directory
                  does. The filename is expected to be of the form:
                  <filename>:<entry_#>. If no experiment index is
                  given, then experiment index 0 (i.e, entry_0) will be
                  assumed by default. A validpath may, e.g, be
                  "test.h5:/entry_0" or jus "test.h5" (default: None)
  --profile        Enable runtime profiling of the analysis. NOTE: This
                  is intended for debugging and investigation of the
                  runtime behavior of an analysis. Enabling profiling
                  entails certain overheads in performance (default: False)
```
--memprofile
Enable runtime profiling of the memory usage of
analysis. NOTE: This is intended for debugging and
investigation of the runtime behavior of an analysis.
Enabling profiling entails certain overheads in
performance. (default: False)

--loglevel {INFO,WARNING,CRITICAL,ERROR,DEBUG,NOTSET}
Specify the level of logging to be used. (default: INFO)

-h, --help
show this help message and exit

ana_0:omsi.analysis.findpeaks.omsi_findpeaks_global:analysis settings:
Analysis settings

--ana_0:integration_width ANA_0:INTEGRATION_WIDTH
The window over which peaks should be integrated
(default: 0.1)

--ana_0:peakheight ANA_0:PEAKHEIGHT
Peak height parameter (default: 2)

--ana_0:slwindow ANA_0:SLWINDOW
Sliding window parameter (default: 100)

--ana_0:smoothwidth ANA_0:SMOOTHWIDTH
Smooth width parameter (default: 3)

ana_0:omsi.analysis.findpeaks.omsi_findpeaks_global:input data:
Input data to be analyzed

--ana_0:msidata ANA_0:MSIDATA
The MSI dataset to be analyzed (default: None)

--ana_0:mzdata ANA_0:MZDATA
The m/z values for the spectra of the MSI dataset
(default: None)

ana_1 : generic:
--ana_1:axis ANA_1:AXIS
optional workflow executor options:
Additional, optional settings for the workflow execution controls

--reduce_memory_usage REDUCE_MEMORY_USAGE
Reduce memory usage by pushing analyses to file each
time they complete, processing dependencies out-of-
core. (default: False)

--synchronize SYNCHRONIZE
Place an MPI-barrier at the beginning of the execution
of the workflow. This can be useful when we require
that all MPI ranks are fully initialized. (default: False)

how to specify ndarray data?
--------------------------
n-dimensional arrays stored in OpenMSI data files may be specified as
input parameters via the following syntax:

-- MSI data: <filename>.h5:/entry_#/data_
-- Analysis data: <filename>.h5:/entry_#/analysis_#/<dataname>
-- Arbitrary dataset: <filename>.h5:<object_path>
E.g. a valid definition may look like: 'test_brain_convert.h5:/entry_0/data_0'
In rare cases we may need to manually define an array (e.g., a mask)
Here we can use standard python syntax, e.g. '[1,2,3,4]' or '[[1, 3], [4, 5]]'
7.1 Building the online documentation

The sources for the sphinx docs are part of the doc/ folder and can be build locally as usual via, e.g., `make html`. This will build the documentation locally only as part of the doc/_build folder.

The online documentation is hosted via GitHub pages and are part of the gh-pages branch. The sphinx docs build scripts are conveniently set up to simply the update of the online documentation as part of the regular development process. However, before we can do this we need to do the following simple setup first:

- `cd` to the your local copy of the BASTet repo (i.e., where the /oms and /doc folder are located)
- `cd ..`
- `mkdir bastet_docs`
- `cd bastet_docs`
- `git clone https://github.com/oruebel/BASTet.git html`
- `git checkout -b gh-pages remotes/origin/ghpages`

Once we have created and setup our `bastet_docs` repo for the gh-pages branch we can now build the online documentation as usual from the doc folder of our development repo via the following commands:

- `make htmlpublic`: Rebuild the html docs locally and then copy the docs to the ../../../bastet_docs repo with the gh-pages branch. This will only make local changes without committing or publishing anything.
- `make latexpdfpublic`: Rebuild the latex and pdf and then copy the pdf to the ../../../bastet_docs repo with the gh-pages branch. This will only make local changes without committing or publishing anything.
- `make updatepublic`: Once you have confirmed that the docs are correct, you can rebuild the html and pdf docs for publications using this command. The command also commits and pushes all changes to the docs back to GitHub for publication, so that they become immediately available online.
The following discusses the specification and use of OMSI mass spectrometry imaging data files.

### 8.1 Data Layout

**Experiment Information**

- `/entry_# (HDF5 group)`: Each `/entry#` groups stores data for a single imaging experiment. Data from multiple related experiments may be stored in different `/entry_#` groups within the same file. To link data from related experiments that are stored in separate HDF5 files, one could create a master-file in which the `/entry_#` are softlinks to the corresponding groups in the external HDF5 files.

  - `/entry_#/experiment_identifier (String dataset)`: For each experiment a user-defined identifier name—which should be unique—is stored. This can be used to search for a particular experiment based on its name.

**MSI Data**

- `/entry_#/data_# (HDF5 dataset)`: Multiple original images may be stored for each imaging experiment. The OMSI format manages the data associated with each dataset in a separate group. For a standard MSI dataset this group contains the following datasets:

  - `/entry_#/data_#/data_# (3D dataset)`: Each data group may contain multiple copies of the same data. This is to allow for optimizations of the data layout for different selective read operations. This is important to enable fast data access to spectra and images via the web. By default the omsi format uses 3D arrays to store the spectral imaging data because:

    - This allows the use of chunking in HDF5 to optimize data access for slicing operations in all dimensions. 1D and 2D layout allow for optimization only for a limited set of selection operations.

    - When using, e.g., a 1D data array to store the data, one would also need to store the resolution and order of the different dimensions to make sure that one can interpreted and read the data correctly.

  - `/entry_#/data_#/format (String dataset)`: Simple string dataset describing the internal layout type of the dataset. While the most common data layout is to store the data in a single 3D array, other data layouts are supported as well, e.g., to enable more efficient storage for sparse MSI datasets.

  - `/entry_#/data_#/mz (1D dataset)`: The m/z values associated with the data.

**Instrument Information**

- `/entry_#/instrument (HDF5 group)`: For each experiment a group with additional data about the instrument is stored. The example shown above defines a name for the instrument but we could define a larger list of optional instrument information that we would like to store here. The CXIDB format allows multiple instruments for each experiment. To assign data uniquely to an instrument the data is stored in the corresponding instrument group. However, to ease access to the data, CXIDB defines soft-links to the data.
Fig. 8.1: Illustration of an example HDF5 file using the OpenMSI data format.
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

in the /entry_# group for the experiment as well. Allowing multiple instruments for a single experiment makes the format very complicated and is unnecessary in most cases. In the OMSI format defined here, one can still store data from related experiments in a single file, simply by creating separate entry_# group.

* /entry_#/instrument/name (String dataset): The name of instrument used

* /entry_#/instrument/mz (1D dataset): The mz data of the instrument

Sample Information

- /entry_#/sample (HDF5 group): For each experiment a group with additional data about the sample used in the experiment is stored. The example shown below only defines a name for the sample but we could define a larger list of optional instrument information that we would like to store here. The proposed format here makes a similar simplification compared to the CXIDB format as in the case of the instrument. For each experiment —represented by a /entry_# group— only a single sample may be used. Data from related samples may be stored in the proposed format in separate /entry_# groups representing different experiments.

* /entry_#/sample/name (String dataset): Name of the sample imaged.

Data Analysis Results

- /entry_#/analysis_# (HDF5 group): Multiple derived analysis results may be stored in the proposed format in analysis_# groups associated with the experiment they were created from. Similar to the experiment a user-defined analysis-identifier string should be given to each analysis to allow searches for analysis results by name. Which data needs to be stored for an analysis will depend on the analysis. The omsi python API specifies some base classes to ease integration of analysis algorithms with the API and the HDF5 data format. Further formalizations may be specified to ease support of specific types of analysis results —e.g., clustering results— via the OpenMSI web-interface.

* /entry_#/analysis_#/analysis_identifier (String dataset): For each analysis a user-defined identifier name —which should be unique— is stored. This can be used to search for a particular experiment based on its name.

* /entry_#/analysis_#/analysis_type (String dataset): String describing the type of analysis. This should be high-level category, e.g., peak_finding_local, peak_finding_global, clustering etc. We will define a set of these categories that should be used. Having high-level categories for different algorithms that store their data in the same fashion will help later on with analyzing and visualizing the results from different algorithms that essentially produce the same output.

* /entry_#/analysis_#/... (Arbitrary HDF5 dataset): In the example shown in Illustration of an example HDF5 file using the OpenMSI data format above, we have an example /entry_0/analysis_0/peakcube. The name for the output datasets from the analysis is currently not restricted. A set of name convention will however be defined by the omsi_analysis associated with the indicated analysis_type to ensure that the data can be handled gracefully. Otherwise, if an unknown analysis type is given then all datasets in the /entry_#/analysis_# group that are not part of the standard are assumed to be analysis datasets.

* /entry_#/analysis_#/parameter (HDF5 Group): Group containing additional datasets with input parameters of the analysis

  * /entry_#/analysis_#/parameter/... (Arbitrary HDF5 datasets): Datasets defining input parameters of the analysis

* /entry_#/analysis_#/dependency (HDF5 Group): Group containing additional datasets specifying dependencies of the analysis.

  * /entry_#/analysis_#/dependency/... (HDF5 Group): Each dependency is defined in a separate group containing the following required datasets

8.1. Data Layout


• /entry_#/analysis_/dependency/.../main_name (String dataset): Path to the HDF5 object the analysis depends on.

• /entry_#/analysis_/dependency/.../parameter_name (String dataset): Name of the analysis parameter that has the dependency.

• /entry_#/analysis_/dependency/.../selection (String dataset): Optional Numpy selection string, indicating the subset of the data used.

* /entry_#/analysis_/runinfo/ (HDF5 Group): Group containing all runtime information data, e.g., start, stop and execution times, system and OS metadata etc.

Attributes
HDF5 attributes are used by the OMSI file format only to store format related information but not to store any data. Currently the following attributes are associated with the different high level groups:

• `omsi_type (String)`: Attribute indicating the omsi_file API object to be used to manage the given group. If the attribute is not present then the API decides which API object to use base on the name conventions described above.

• `version (String)`: Attribute indicating the version of the API class that should be used to represent this group.

8.2 Accessing OMSI data files

The `omsi.dataformat.omsi_file` module provides a convenient python-based API for reading and writing OMSI data files. The class also provides a convenient function for generating a XML-format XDMF header for the OMSI HDF5 file. Using the XDMF header file, the HDF5 data can be loaded in VisIt using VisIt’s XDMF file reader. OMSI data files are valid HDF5 data files and can be accessed via the standard HDF5 libraries.

• **C/C++:**
  - More information about HDF5 can be found here: [http://www.hdfgroup.org/HDF5/](http://www.hdfgroup.org/HDF5/)

• **Python:**
  - H5Py is a python interface to the HDF5 library. More detailed information can be found here: [http://h5py.alfven.org/docs-2.0/](http://h5py.alfven.org/docs-2.0/)

• **MATLAB:**
  - MATLAB provides both high-level and low-level access functions to HDF5. For more details see [http://www.mathworks.com/help/techdoc/ref/hdf5.html](http://www.mathworks.com/help/techdoc/ref/hdf5.html)
  - Simple example using the high-level API:

```python
file='~/Data/Imaging/DoubleV.hdf5
h5disp(file)
mz=h5read(file,'/entry_0/instrument/mz');
[mx]=min(abs(mz-746.22))
tic
y=h5read(file,'/entry_0/data_0',[mx 1 1],[1 250 160]);
toc
imagesc(squeeze(y))
axis equal
axis tight
```

- Simple example using the low-level API:
file='~/Data/Imaging/DoubleV.hdf5'
plist = 'H5P_DEFAULT';
fid = H5F.open(file);
gid = H5G.open(fid,'/entry_0');
dset_id = H5D.open(fid,'/entry_0/data_0');
dims = [160 250 1];
offset = [0 0 mx]
block = dims;
mem_space_id = H5S.create_simple(3,dims,[]);
file_space_id = H5D.get_space(dset_id);
H5S.select_hyperslab(file_space_id,'H5S_SELECT_SET',offset,[],[],block);
tic
data = H5D.read(dset_id,'H5ML_DEFAULT',mem_space_id,file_space_id,plist);
toc
H5D.close(dset_id);
H5F.close(fid);
data=squeeze(data);
imagesc(data)
axis equal
axis tight

• Using HDF5 at NERSC
  – Overview of python at NERSC: http://www.nersc.gov/users/software/development-tools/python-tools/
  – HDF5 modules are installed on most machines at NERSC:
    module load hdf5
    module load python
    python
    >>> import h5py
    >>> import numpy

8.3 Convert Mass Spectrometry Imaging Data to OMSI (HDF5) format

See section Converting Files at NERSC and Making them Accessible for details.

• omsi.tools.convertToOMSI: This python script, which is available via the OMSI software toolkit, provides functionality for converting img files to HDF5. The script takes a single or multiple img files as input and writes them to a single HDF5 file. The data of each img file is stored in a separate /entry#/data_# object. The script also supports execution of a number of different analysis, such as, peak finding or nmf, directly during the data conversion. For up-to-date information about the usage of the script see python imgToHDF5 --help. A summary of the main command-line options of the tool are available below.

• omsi.dataformat.omsi_file: Module containing a set of python class for reading and writing HDF5 data files for the proposed OMSI HDF5 data layout.

• omsi.dataformat.img_file: Simple python class for reading img data files.

• omsi.dataformat.bruckerflex_file: Simple python class for reading bruckerflex files.
Analysis and visualization of mass spectrometry imaging (MSI) data is often based on selected subsets of the data, e.g., single or multiple spectra or m/z data slices. It is, therefore, crucial that we are able to quickly access select subsets of the data. In the context of web-based applications — such as the OpenMSI Viewer — this is particularly crucial in order to enable interactive data analysis and to provide a satisfactory user experience. The tests described here focus on characterizing and optimizing the performance of data access operations performed in serial on OMSI HDF5 files. While we here focus on the performance of serial data accesses, the optimizations described here are fundamental to optimizing parallel data access as well.

In the following, we first identify a select set of target compute platforms (Section 9.1 Test Platforms) and define a set of representative use cases in order to evaluate the performance of different data layouts (Section 9.2 Test Cases). We then discuss the basic layout of the MSI data (Section 9.3 Dataset Layout) and establish the baseline performance using the default contiguous data layout (Section 9.4 Baseline Performance). Afterwards, we explore further optimization of the data layout using HDF5’s data chunking (Section 9.5 Chunking: Part I) and data compression (Section 9.6 Compression) capabilities. We conclude this study with a discussion of lessons-learned in Section 9.7 Discussion.

9.1 Test Platforms

All tests were performed on two main compute systems: i) login node of hopper.nersc.gov (short hopper) and ii) portal-auth.nersc.gov (short portal). On hopper we utilized the LUSTRE-based /scratch file system, as well as the global GPFS-based /project file system. On portal we can only access the /project file system. We chose these systems because: i) hopper is our candidate system for performing large-scale parallel analysis of MSI data and ii) portal is our target system for providing web-based access to MSI data.

9.1.1 hopper.nersc.gov

The hopper system has 12 login nodes with 4 quad-core AMD 2.4 GHz Opteron 8378 processors (16 cores total) each on 8 of the login nodes and 4 8-core AMD 2.0 GHz Opteron 6128 processors (32 cores total) each on 4 of the login nodes. Each login node has 128 GB of memory. The login nodes are external to the main Cray XE6 system. All tests were performed using hopper login nodes.

Scratch: There are two Lustre file systems on hopper — mounted as /scratch and /scratch2 (in the following we use /scratch2) — with the following setup:

- 13 LSI 7900 disk controllers (Each disk controller is served by 2 I/O Object Storage Servers (OSSs))
- Each OSS host 6 OSTs (Object Storage Target) (simplified speaking a software abstraction of a physical disk)
- Fiber Channel 8 connectivity from OSSs to the LSI disk controllers
- Infiniband connects the Lustre router nodes in the 3d torus through a QDR switch to the OSSs
• In total each /scratch file system has 156 OSTs which is the lowest layer with which users need to interact. When a file is created in /scratch it is by default “striped” or split across two different OSTs. Striping is a technique to increase I/O performance. Instead of writing to a single disk, striping to two disks allows the user to potentially double read and write bandwidth. In the following experiments we use the default stripping settings but depending on file size, larger stripe settings may be advantageous. Using the /scratch file system for temporary storage of MSI data files can be advantageous when performing complex I/O intensive analysis.

Global /project: This is a large (1606 TB), permanent, medium-performance GPFS-based file system. We utilized the /project file system in the context of the OpenMSI project for permanent storage of MSI data.

9.1.2 portal-auth.nersc.gov

The portal system is used at NERSC for any data services that require public access from the Internet (such as Science Gateways) and as such also hosts the OpenMSI webpage and science gateway. The system consists of 2 quad-core AMD Opteron 2378 processors (8 cores total) with 24GB of memory. This system has only access to the /project file system.

9.2 Test Cases

In order to evaluate the performance of different data layouts, we designed a set of test-cases modeling the most common data access patterns in the analysis of MSI data. One particular focus of this study is to optimize the performance of the file format for web-based access to the data required for OpenMSI’s online data viewing, analysis and exploration functionality. In this context it is most important that we are able to quickly access select subsets of the data, in particular, image slices, spectra or subcubes of the data. These type of data access patterns, however, are very common also for a large range of data analyses, e.g., peak finding on individual spectra, data clustering and many others. In contrast to analysis performed with direct access to the compute system, the abilities for data caching are typically much more limited in a web-based setting due to the fact that: i) http accesses are stateless, i.e., a file is typically reopend for each incoming request and closed again afterwards and ii) access patterns to the data are much more irregular with multiple users working with different datasets and/or different subsets of the data at the same time. While the median performance for repeated data selection operations on the same open file is often very important for data analysis, in a web-based setting the maximum time for the first access to the data is often much more important. In the following we report for each selection test case the median time (indicating the sustained performance on an open file) and the maximum time (indicating the selection performance after the first opening of the file). We usually repeat each selection test case 50 times for each data layout using randomized selection parameters.

9.2.1 Case 1: m/z Slice Selection

This test case models the selection of a series of z-slices of the data (i.e., slices in a mass range), and extracts a set of consecutive, full images of the data. This type of operation is required in the OpenMSI viewer when updating channels in the image viewer itself. It is also a common operation in many other analyses, e.g, when analyzing the distribution for a particular peak across the image.

- **Randomized Selection Parameters:** zmin
- **Dependent Selection Parameters:** zmax = zmin+25
- **Extracted Dataset:** 100 \times 100 \times 25 = 250,000 records = 500,000 bytes = 0.5MB

9.2.2 Case 2: Spectra Selection

This test case models the selection of a 5 \times 5 set of full spectra. In the OpenMSI viewer, access to single and multiple neighboring spectra is required when updating the spectrum plot’s. This is also a typical operation for many analyses
that operate on a per spectrum basis, e.g., peak finding for a single spectrum.

- **Randomized Selection Parameters**: \(x_{\text{min}}, y_{\text{min}}\)
- **Dependent Selection Parameters**: \(x_{\text{max}} = x_{\text{min}} + 5, y_{\text{max}} = y_{\text{min}} + 5\)
- **Extracted Dataset**: \(5 \times 5 \times 100,000 = 200,000\) records = \(2,500,000\) bytes = \(5\) MB

### 9.2.3 Case 3: 3D Subcube Selection

This selection models the general access to consecutive sub-pieces of the data, e.g., when accessing data from a particular spatial region of the data related to a particular set of m/z data values. This type of operation is required, e.g., when analyzing the data of a cluster of pixels with a particular set of peaks of interest.

- **Randomized Selection Parameters**: \(x_{\text{min}}, y_{\text{min}}, z_{\text{min}}\)
- **Dependent Selection Parameters**: \(x_{\text{max}} = x_{\text{min}} + 5, y_{\text{max}} = y_{\text{min}} + 5, z_{\text{max}} = z_{\text{min}} + 1000\)
- **Extracted Dataset**: \(5 \times 5 \times 1,000 = 25,000\) records = \(50,000\) bytes = \(0.05\) MB

### 9.2.4 Case 4: Data Write

As described above, the aim of this study is to optimize the performance of selective data read operations (termed hyperslap selections in HDF5). In contrast to the data read, data write is a one-time cost during the file conversion step and is, therefore, less critical to the operation of OpenMSI. A reduced write performance may, therefore, be acceptable in lieu of an increase in read performance as long as an acceptable write performance is maintained. During selection performance tests, the data write is repeated only 3 times for each data layout (i.e., once for each of the three selection test cases). For selected cases (indicated in the plot titles) we ran dedicated data write tests with 10 repeats. We here typically report the average times for data write.

### 9.2.5 Case 5: File Size

The size of data files is important to this study as different file layouts may have different space requirements (e.g., due to padding and additional metadata). While reduction of the size of data files is not the main objective of this work, it is important to avoid unnecessary overheads in file size and, hence, storage cost. The size of files reported in this study have been determined using the Python command `os.stat( filename ).st_size`.

### 9.2.6 Test Data

For this study we use a \(100 \times 100 \times 100\), 000 test dataset. The dataset is stored as a 3D array of UInt (16bit) data values using the OMSI HDF5 format described in Chapter OMSI Data Format. Data write and hyperslap performance are independent of the data values being written/read, so to test the baseline write-performance, we simply assign to each data element the index of the corresponding data chunk. For test cases that utilize data compression, we use a donor MSI data file to fill the file with realistic data. We may replicate data from the donor file in case that the testfile is larger than the donor file. In case that a donor file is used, we read the donor data into memory prior to writing of the test dataset. For each test case (i.e., data layout + selection case) we generate a new test data file to reduce/eliminate effects of data caching. The newly generated file is then opened and the current selection (i.e., hyperslap selection) is repeated 50 times using randomized selection parameters.

Raw data files used in this study are:

- **Dataset A**: (default donor file)
  - **Name**: 11042008_NIMS.h5
Dimensions: 227 × 108 × 63,378

Raw Data Size: 3.2GB

File Size: 3.3GB (including results from global peak finding and nmf)

• Dataset B:
  
  Name: 2012_0403_KBL_platename.h5 (DoubleV)
  
  Dimensions: 160 × 250 × 116,152
  
  Raw Data Size: 9.3GB
  
  File Size: 9.5GB (including results from global peak finding and nmf)

9.3 Dataset Layout

A single (2D) MSI dataset defines a 3D data volume with the spatial coordinates $x$, $y$ and the $m/z$ (mass) as third dimension ($z$). In raw block-of-uint format (e.g., in the IMG format) the data is often stored in a 1D linearized fashion: $\textit{spectrurm}_{0,0}$, $\textit{spectrurm}_{0,1}$, $...$, $\textit{spectrurm}_{l,m}$. While such a layout is well-suited for accessing single full spectra, access to single image z-slices requires $l \times m$ seek operations and traversal of the complete dataset (with $l$, $m$ being the number of pixel in $x$, $y$, respectively). Selection of spectra and z-slices of the data are orthogonal selection operations, i.e., a 1D data layout can always just optimize one of the two access operations but not both. Similarly, a 2D data layout can be defined to enable easy access to full spectra as well as full z-slices, but does not easily support to optimize access to 3D subsets of the data. We, therefore, store MSI data as a 3D array in HDF5 to: i) be able to optimize and find a good performance compromise for selection of spectra, z-slices as well as 3D subcubes of the data and ii) because the 3D array reflects the true dimensionality of the data.

HDF5 can represent array datasets with as many as 32 dimensions. However, in the file the data is linearized in order to store it as part of the 1-dimensional stream of data that is the low-level file. The data layout determines in which way the multidimensional dataset is mapped to the serial file. The simplest way to accomplish this is to flatten the dataset (similar to how arrays are stored in memory) and to store the entire dataset into a monolithic block on disk. We here use this, so-called, contiguous layout as baseline for our performance tests (see Section Baseline Performance).

Chunking provides an alternative to the contiguous layout. In contrast to storing the data in a single block in the HDF5 file, using chunking the data is split into multiple chunks. Each chunk of a dataset is allocated separately and stored at independent locations throughout the HDF5 file. The chunks of a dataset can then be read/written independently, enabling independent parallel I/O and potentially improving performance when operating on a subset of the dataset. Data chunks may be stored in arbitrary order and position within the HDF5 file. HDF5 uses a B-tree to map a chunks N-dimensional address to a physical file addresses. The size of the B-tree directly depends on the number of chunks allocated for a dataset. The more chunks are allocated for a dataset: i) the larger overhead for traversal of the B-tree, ii) the higher the potential contention for the metadata cache, and iii) the larger the number of I/O operations. An introduction to data chunking in HDF5 is provided at http://www.hdfgroup.org/HDF5/doc/Advanced/Chunking/. The performance of different chunking strategies for storing MSI data is evaluated in Section Chunking: Part 1.

Use of chunking also enables the use of HDF5 I/O filters. We here investigate the use of compression filters. I/O filters are applied in HDF5 to each chunk individually, and entire chunks are processed at once. I/O filters enabled for a dataset are executed every time a chunk is loaded or flushed to disk. Choosing proper settings for the chunking (and chunk cache) are, therefore, critical for the performance of filtered datasets. The potential use of gzip compression for improving file size and hyperslab selection of MSI datasets is evaluated in Section Compression.

Chunking and HDF5 I/O filters (e.g., data compression) are implemented transparently in HDF5, i.e, the API functions for reading/writing chunked/compressed datasets are the same ones used to read/write datasets with a uncompressed, contiguous layout. The layout (i.e., chunking scheme and compression options) is defined via a single function call to set up the layout on a property list before the dataset is created.
9.3.1 Baseline Performance

The goal of this section is to establish a baseline for the performance of the basic HDF5 contiguous data layout (i.e., without chunking). The baseline performance for the three selection test cases are shown in Figures i) Baseline performance for the slice selection test case, ii) Baseline performance for the spectra selection test case, and iii) Baseline performance for the subcube selection test. The bar plots show the minimum (blue), median (blue+red), average (blue+red+green), and maximum (blue+red+green+lilac) times for retrieving the selected data. We observe that hopper using /scratch provides much better performance for selection of spectra and 3D subcubes of the data. For the z-slice selection we observe that hopper achieves good median and average selection performance, whereas the performance of the z-slice selection on portal is generally poor. In all cases, we observe poor worst-case (maximum) times for the z-slice selection case.

For serial data write performance we observe that /project provides better write performance. However, for parallel write operations, the LUSTRE-based /scratch file system is expected to outperform /project.

![Baseline Times for selection 25 m/z slices of the data](image1)

Fig. 9.1: Baseline performance for the slice selection test case

![Baseline times for selecting a 5x5 subset of full spectra](image2)

Fig. 9.2: Baseline performance for the spectra selection test case
Fig. 9.3: Baseline performance for the subcube selection test

Fig. 9.4: Baseline performance for data write
9.4 Chunking: Part 1

Use of a chunked data layout has many potential advantages. In particular, use of chunking enables independent data I/O operations on individual chunks of the data so that chunking: i) can reduce the amount of data that needs to be read during hyperslab selections, ii) enables parallel independent I/O on a single file, and iii) enables the use of data compression (discussed later in Section Compression). The goal of this section is to evaluate the use of chunking to improve I/O performance. Due to the amount of additional metadata and overhead associated with finding chunks, one should avoid the use of too small chunks. At the same time, use of too large chunks should be avoided, because the entire chunk must be read from disk (and decompressed) before performing any operations. When operating on small subsets of the data (and if the cache is too small to hold the chunk), the use of too large chunks can result in large performance penalties. In addition, if the chunk is too large to be held in memory, the operating system may have to page memory to disk, slowing down the entire system.¹

Choosing a good chunking strategy for MSI data is complicated because: i) the data has a very unconventional shape, with the m/z dimension being three to four orders of magnitude larger than the spatial x/y dimensions and ii) orthogonal data access operations (access to spectra vs. z-slices) are required with good first-time-access performance.

To account for these properties we use odd chunk sizes of \( m \times m \times n \) with \( n >> m \). Finding a good compromise for choosing a good chunking is challenging. Larger chunk sizes \( m \) in \( x, y \) are expected to improve z-slice selections but also increase the overhead for spectra selections. Similarly, large chunk sizes \( n \) in \( z \) (m/z) are expected to improve spectra selections while increasing the overhead for z-slice selections. The goal of this first set of experiments is to find a chunking that provides a good compromise in performance for all three selection test cases.

In the following we compare the performance of a range of different chunking strategies of the form \( m \times m \times n \) with \( m \in \{1, 2, 4, 8, 16, 32\} \) and \( n \in \{128, 256, 512, 1024, 2048, 4096, 8192\} \) using hopper using /scratch. We first evaluate the effects of the different data layouts on file size (Section File Size) and write performance (Section Data Write). We then compare the performance for performing the three selection test cases (Section Selection Performance). We conclude this chunked layout study with an evaluation of the overall performance of the different chunked data layouts to identify the best-performing data layouts (Section Selection Performance).

9.4.1 File Size

The use of chunking affects the size of data files in two main ways. First, storing the additional metadata required for chunking —such as the B-tree used for indexing of data chunks— increases file size. Second, the use of chunking may result in allocation of additional empty data (padding) in case that the chunks do not align with the data. This can result in substantial data overahes. A simple example illustrates this problem. When storing a simple 1D dataset with 101 elements using a chunk size of 100, then we need to allocated two chunks, one chunk to store the first 100 elements and a second chunk to store the last element. In this case we allocated space for 200 elements in order to store 101 elements, nearly twice the amount of storage needed for the raw data. For multi-dimensional data arrays —here 3D— the storage overheads due to padding can increase even faster. It is, therefore, important that we consider the potential storage overhead when evaluating the use of data chunking.

Figure File sizes using different chunking strategies illustrates the effects of chunking on the size of data files. The baseline curve indicates the file size using a contiguous data layout. We observe that the file with a chunking of \( 1 \times 1 \times 128 \) is much larger than the other files with a \( 1 \times 1 \times n \) chunked layout. No padding is applied in the spatial dimensions \( x, y \). When using a \( z \) chunk size of 128, 782 chunks are required per spectrum, resulting in a total of 7,820,000 chunks. Due to padding in the \( z \) dimension, 96 \( 100 \times 100 \) slices remain empty. However, this accounts for only 100 * 100 * 96 * 2Bytes = 1,920,000Bytes = 1.92MB. In comparison, the \( 1 \times 1 \times 2048 \) dataset is much smaller while requiring a much larger \( z \) padding of 352 slices (i.e, \( \approx 7.04MB \)). The reason for the larger file size for the \( 1 \times 1 \times 128 \) chunking illustrates the large overhead for storing the metadata required for the large number of chunks.

We also observe that the file size increases significantly when using chunk sizes in \( x, y \) of \( 8 \times 8 \times n \) or larger. This behavior is due to the padding required in the spatial dimensions. For example, when using a chunking of

¹ See also http://www.hdfgroup.org/HDF5/doc/Advanced/Chunking/
32 × 32 × n we requires 4 chunks in the x and y dimension (i.e., 4 × 32 = 128 elements). This means, in order to store the 100 × 100 × 100,000 test dataset, we allocate space for at least 128 × 128 × 100,000 records (additional padding may be required in the z dimension). This means that we allocate at least an additional amount of space of (28 × 128 × 100,000) + (28 × 100 × 100,000) records = 638, 400, 000 × 2 Bytes = 1, 276, 800, 000 Bytes = 1276.8 MB. This example illustrates that a bad choice for the chunking can result in substantially larger data files. Since in the case of MSI data, the x, y dimensions of the data are much smaller than the z (mz, mass) dimension, it is important that we keep the padding required in x and y as small as possible, whereas padding in the z dimension typically has a much smaller effect on the size of the data.

### 9.4.2 Data Write

Traditionally, MSI data is often written one-spectrum-at-a-time. Figure Write performance using one-spectrum-at-a-time I/O using different chunk sizes (hopper using /scratch) illustrates the write performance for the different data layouts using a one-spectrum-at-a-time write strategy. It is not surprising that we observe a significant decrease in performance with increasing chunk sizes m in the x and y dimensions, as each data chunk is modified m × m times. For chunkings of m × m × 32 with m ∈ {1024, 2048, 4096, 8192} the write performance improves possibly due to higher HDF5 chunk-cache hit rates. The write performance data points for data layouts with a chunking of 128 × 128 × 32, 256 × 256 × 32, and 512 × 512 × 32 are missing as we terminated the tests due to the very poor one-spectrum-at-a-time write performance in those cases.

To achieve optimal data write performance, it is important that we reduce the number of times each chunk is modified. Figure Write performance using different data write strategies and chunk sizes (hopper using /project) compares the write performance for m × m × 2048 chunked data layout using a one-spectrum-at-a-time, m × m-spectra-at-a-time, and chunk-at-a-time data write strategy. Using the latter two strategies ensures that each chunk is modified only once. We observe that the chunk-at-a-time write strategy quickly outperforms the other write strategies as the chunk size increases (and the total number of chunks decreases). It is not surprising that the contiguous baseline layout outperforms the chunked layouts in a serial setting. However, the chunked layouts efficiently support parallel data write operations. Using a chunk-at-a-time write strategy, independent parallel tasks can be utilized to write the different chunks.
Fig. 9.6: Write performance using one-spectrum-at-a-time I/O using different chunk sizes (hopper using /scratch)

Fig. 9.7: Write performance using different data write strategies and chunk sizes (hopper using /project)
9.4.3 Selection Performance

In this section we evaluate the selection performance of the different chunked data layouts for the three selection test cases: i) selection of a random set of 25 consecutive z-slices, ii) selection of a random $5 \times 5$ set of full spectra, and iii) selection of a random $5 \times 5 \times 1000$ subcube of the data. Figure Performance results for z-slice selection using varying chunk sizes (hopper using /scratch) shows the results for the z-slice hyperslap selection. We observe that data layouts with a chunking of $m \times m \times n$ with $m \in \{4, 8, 16\}$ and $n \in \{128, 256, 512, 1024, 2048, 4096\}$ show the best z-slice selection performance. For the mentioned chunking strategies we observe in general best performance for larger values in $m$ and smaller values of $n$. This behavior is likely due to the reduced amount of data and number of chunks that need to be loaded to fulfill the selection of complete slices in $z$.

For the spectra selections (see Figure Performance results for spectra selection using varying chunk sizes (hopper using /scratch)) we observe in most cases a decrease in the median performance compared to the baseline contiguous data layout. This behavior is likely due to the fact that the data is flattened in a z-column order in the contiguous layout, so that a full spectrum can be read via a single seek and contiguous read operation. In contrast, using a chunked data layout requires for the test dataset, loading $100,000/n$ chunks. However, we observe that data layouts with a chunk size in $z$ of 1024 and larger, still provide good performance for the selection of full spectra. In this case, small chunk sizes in $x, y$ of 2 or 4, work well for the test case of loading a $5 \times 5$ set of spectra, as the number of chunks that need to be loaded remains constant. For data layouts with a $x, y$ chunking of $8 \times 8$, the $5 \times 5$ spectra selection can fit into a single $x,y$ chunk, however, it is still likely that the $5 \times 5$ selection crosses multiple chunk boarders, requiring the load of a large number of chunks. For $x, y$ chunk sizes of 16 or 32, the $5 \times 5$ spectra selection is more likely to fit in a single chunk in $x,y$, explaining the better performance of those data layouts.

Figure Performance results for 3D subcube selection using varying chunk sizes (hopper using /scratch) summarizes the results for the selection of a $5 \times 5 \times 1000$ data subcube. We observe that using chunking generally improves the performance of the selection. In particular, using chunking decreases the time for initial data access compared to the baseline contiguous data layout.

Bars shown transparently indicate that the bars exceed the maximum value shown in the plot. The real value for those bars are indicated via additional text labels.
Fig. 9.9: Performance results for spectra selection using varying chunk sizes (hopper using /scratch)

Fig. 9.10: Performance results for 3D subcube selection using varying chunk sizes (hopper using /scratch)
When comparing the selection performance plots, we observe that — even though the amount of data that is retrieved is only 0.5MB in z-slice selection case compared to 5MB in the spectra selection case—the performance for z-slice selection case is generally lower than for the spectra selection case. The reason for this behavior is that while the z-slice selection returns less data, the number of I/O (seek) operations required and the amount of data that needs to be loaded to fulfill the selection is larger. E.g., using a chunking of $4 \times 4 \times 2048$ each chunk requires $4 \times 4 \times 2048 \times 2\text{Byte} = 65536\text{Byte} = 64\text{kB}$. To retrieve a $5 \times 5$ set of spectra, HDF5 needs to load $2 \times 2 \times 49$ Chunks $= 196 \times 64\text{ kB} = 12.25\text{ MB}$, whereas in order retrieve 25 z-slices (without crossing a z-chunk boundary), HDF5 needs to load $25 \times 25$ Chunks $= 625\text{ Chunks} = 39.0625\text{ MB}$, i.e, more than three times the data (and that even though the $4 \times 4$ x,y chunking does not align well with the $5 \times 5$ spectra selection). Similary, in the contiguous data layout, we can load a single spectrum using a single seek and contiguous read operation, whereas in order to load complete a z-slice we requires $m \times m$ ($= 100 \times 100 = 10,000$ for the test data) seek and load operations.

### 9.4.4 Summary

To illustrate the overall performance of the different dataset layouts and to identify the “best” layouts, we define the following set of minimum performance criteria a data layout should fulfill:

- The median time for the z-slice selection test case should be $<0.1\text{ s}$
- The median time for the spectra selection test case should be $<0.05\text{ s}$
- The median time for the 3D subcube selection test case should be $<0.002\text{ s}$
- The total file size should be $< 2100\text{ MB}$ (limiting the overhead in the test case to a maximum of $\approx 200\text{MB}$)
- (We do not take the write performance results shown in Figure Write performance using one-spectrum-at-a-time I/O using different chunk sizes (hopper using /scratch) into account in the total score here as a chunk-at-a-time write strategy likely improve the write performance significantly)

Based on these criteria we can determine an overall performance score by evaluating how many of the criteria a particular data layout fulfills (with $4=$best (passes all criteria) and $0=$worst (does not pass any of the criteria)). We observe a cluster of 8 data layouts that satisfy the four performance conditions.

![Summary Performance Score](image)

Fig. 9.11: Summary performance score using various different chunk sizes (hopper using /scratch)
Figure **Summary performance score using various different chunk sizes (hopper using /scratch)** summarizes the performance scores for the different data layouts. Bars with the maximum score of 4 are plotted opaque whereas all other bars are plotted transparently. We observe a cluster of 8 layouts with a performance score of 4. While the overall performance score used here is simple in nature, it illustrates well which data layouts achieve overall the best performance. Overall, the performance experiments indicate that the largest z chunk size for which we observe good performance in our experiments across a large range of x/y chunk sizes is 2048. We, therefore, chose a z chunk size of 2048 for further experiments.

So far our experiments have focused on hopper using the /scratch file system. In order to evaluate the performance of the permanent data storage system /project and the web-hosting system portal we performed a series of follow-up tests using a fixed z-chunking of 2048 on portal as well as on hopper using /project (more details are provided in the next section). We generally observed that the /project file system provided better serial write performance while the overall performance for selection was not as good as for /scratch. While the performance of ‘‘hopper using /project was still acceptable, the performance (and in particular the worst-case maximum times) were poor using /portal (see, e.g. Figure Performance results for z-slice selection (portal using compression) shown later). We, therefore, next extended our evaluation to also include data compression as an option to possibly improve data I/O performance and storage requirements.

### 9.5 Chunking: Part 2

In part 1 of the chunking study we were interesting in finding chunked data layouts that provide a good compromise for all three selection use-cases. In part 2 we focus on chunked data layouts that are designed to optimize single selection operations.

#### 9.5.1 Image-aligned Chunking

Figure **Comparison of the performance of the default chunking with a chunked layout that is better-aligned with the ion-images to improve the performance the z-slice selection.** shows the performance of the z-slice selection in the OpenMSI client when optimizing the chunked data layout to align with the selection of m/z images. Using an image-aligned chunking of $50 \times 80 \times 100$, only 10 chunks containing 100 image slice need to be read in order to retrieve a single image slice, whereas, using the default chunking of $4 \times 4 \times 2048$, HDF5 needs to load 2,520 chunks containing 2048 image slices. The figure shows the performance we achieve in the OpenMSI client using the same data stored without compression using the two different chunked layouts. Using the default chunking it takes $\approx 20s$ to compute the ion-images compared to just $\approx 0.5s$ using the image-aligned chunking. However, not surprisingly, the performance of the orthogonal operation of selecting full spectra decreases significantly (here from 45ms to 8.1s) when optimizing the chunking to improve the selection of z-slices.

Further improvement in performance could be achieved by using a (m/z, x, y) or (m/z, y, x) layout of the data rather than the common (x, y, m/z). By transposing the image cube (making m/z the first dimension of the 3D cube) the data will be linearized on disk as $\text{ionimage}_0, \text{ionimage}_1, \ldots, \text{ionimage}_n$ rather than $\text{spectrum}_{0,0}, \text{spectrum}_{0,1}, \ldots, \text{spectrum}_{m,0}$. Linearizing the data in image order improves locality of data and reduces the number of seek operations required when loading ion-image. Currently, reordering of data dimensions is not yet supported by HDF5 as a transparent data layout optimization, but rather needs to be performed manually by the user. To remain ease of usability of the OpenMSI file format we, therefore, chose to use a consistent ordering of dimensions in all cases.

### 9.6 Compression

The primary goal of data compression is to reduce the size of data by providing a more compact encoding of the data. In many cases, compression is used as means to reduce the size of data stored on disk. However, while additional compute time overheads are incurred due to the time required for compression/decompression of the data during
write/read, compression may also improve the read and/or write performance, as less data needs to be transferred and/or written to disk. This is in particular the case in I/O bound systems (e.g., due to network bottlenecks etc.).

Use of data compression in HDF5 relies on the use of chunking and, hence, shares the same overheads and advantages and disadvantages. Compression is applied to each chunk of the data. The overall compression ratio achieved, therefore, inherently relies on the use of a good chunking strategy. Data compression is implemented via I/O filters in HDF5, which are applied transparently during data read and write operations, i.e., after enabling compression when generating the dataset, data read/write operations are performed using the same API calls whether the data is stored in raw or compressed form.

HDF5/h5py typically provides three main compression algorithms: i) gzip, standard HDF5 deflate compression available with most HDF5 installations, ii) szip, third-party compression algorithm optionally available with HDF5 (i.e., it may not be available on all systems), iii) LZF is a stand-alone compression filter for HDF5 available via h5py but may not be available in many other standard (non-Python) installations of HDF5. In the context of OpenMSI it is important that we are able to transfer and use data at different institutes, compute systems and using a larger range of API’s for accessing HDF5 data (e.g., matlab, HDF5 C API, h5py etc.). We, therefore, chose the standard gzip compression filter as it is typically available with most systems (in contrast to LZF and szip). The gzip filter provides and additional aggression parameter. The aggression parameter is a number between [0, 9] to indicate the trade-off between speed and compression ratio (zero is fastest, nine is best ratio). Unless indicated otherwise, we here generally set the aggression parameter to 4 for all tests to achieve a balance of compression ratio and speed.

9.6.1 Compression Ratio

Using a chunking of $4 \times 4 \times 2,048$, we achieve for dataset $A$ a compression ratio of $\approx 2.9$, reducing the data from 3.3GB (including $\approx 100$ MB of data for nmf and global peak-finding) to 1.2GB (while only the MSI data is stored in compressed form). Using the same setup, we achieve for dataset $B$ a compression ratio of $\approx 6.3$, reducing the dataset from 9.5GB to 1.5GB (again with nmf and global peak-finding results included in the file and stored uncompressed in both cases). An overview of the file sizes using compression and compression ratios achieved using the $100 \times 100 \times 100,000$ test dataset (using dataset $A$ as donor MSI dataset) are summarized in Figure Compression ratios using different chunk sizes. The compression ratios we achieve are on the order of 2.9 to 3.8 in all cases (comparable to the compression ratio we have seen for the donor dataset $A$).
Fig. 9.13: Compression ratios using different chunk sizes.

Fig. 9.14: Compression performance on a select set of real MSI datasets.


9.6.2 Data Write

Figure Serial write performance using compression compares the write performance results with and without using compression for data layouts with a chunking of $m \times m \times 2048$ with $m \in [1, 10]$. As expected, we observe a general decrease in the write performance when using compression. As mentioned earlier, when using compression it is important that we use a chunk-aligned data write strategy as the compression filter needs to be executed each time a chunk is loaded and/or modified. We, therefore, use the chunk-at-a-time write strategy when writing HDF5 datasets with compression enabled.

![Chunk-at-a-time write performance: hopper using /project](image)

Fig. 9.15: Serial write performance using compression

9.6.3 Selection Performance

In this section we evaluate the selection performance of the different $m \times m \times 2048$ chunked data layouts (with $m \in [1, 10]$) for portal and hopper using /project and /scratch. For portal we observe that using compression can significantly improve the median selection time for the z-slice selection test case (see Figure Performance results for z-slice selection (portal using compression)). In particular, using compression significantly reduces and stabilizes the worst-case maximum time for selecting z-slices of the data. This is especially important in the context of a web-based application, such as, OpenMSI’s online data viewer. For the other two selection test cases we observe that we can achieve similar performance for the spectra and 3D subcube selection test cases on portal with and without compression (see Figures Performance results for spectra selection (portal using compression) and Performance results for 3D subcube selection (portal using compression)).

The selection performance results for hopper using /project and /scratch and with and without compression are shown in Figures: i) Performance results for z-slice selection (hopper using compression), ii) Performance results for spectra selection (hopper using compression), and Performance results for subcube selection (hopper using compression). As baseline we use the performance of the contiguous data layout on hopper using /project. In contrast to portal, we observe on hopper a general decrease (of up to 1 order of magnitude) in the selection performance when using compression compared to when storing the data in raw, uncompressed form. Compared to the

---

Author comment: Depending on when the test were run we have seen significant (approximately 1 order of magnitude) differences in the median selection times on portal, however, the maximum appeared to not improve between different reruns of the experiments. Using compression, the results on portal have been much more stable and, in particular, the maximum times were much better. This indicates that: i) system load, on a highly utilized system like portal, has a significant impact on the performance and ii) compression can significantly improve performance the selection performance on I/O bound systems and stabilized the performance results.
baseline contiguous data layout we, however, still observe an improvement in many cases even when using compression. Generally it appears that the use of compression may have stronger negative effect on the selection performance when operating on the /scratch filesystem then when using /project.

Fig. 9.16: Performance results for z-slice selection (portal using compression)

Fig. 9.17: Performance results for spectra selection (portal using compression)

9.6.4 Aggression Parameter Study

So far we have focused on the performance using gzip with aggression = 4, assuming that a medium aggression value provides a good balance of compression ratio and speed. The goal of this section is to determine the influence

Author comment: Note, /scratch is a parallel, LUSTRE-based file system, whereas /project is based on GPFS. Also hopper may have a higher bandwidth connection to the /scratch file system than /project. Overall, accesses to /project is likely to be more I/O bound than access to /scratch.

The aggression parameter is a number between [0, 9] to indicate the tradeoff between speed and compression ratio (0=fastest, 9=best ratio).

9.6. Compression 67
Fig. 9.18: Performance results for 3D subcube selection (portal using compression)

Fig. 9.19: Performance results for z-slice selection (hopper using compression)
Median time for selecting 5x5 subset of complete spectra on hopper with error bars indicate the maximum time.

Fig. 9.20: Performance results for spectra selection (hopper using compression)

Median time for selecting 5x5x1000 subcube on hopper with error bars indicate the maximum time.

Fig. 9.21: Performance results for subcube selection (hopper using compression)
of the \textit{aggression} setting on the overall performance. Figure \textit{File size and write performance using varying gzip aggression settings} shows that the data compression we achieve for the test $100 \times 100 \times 100,000$ dataset (using dataset A as donor dataset) are comparable for all $\text{aggression} \in [1, 8]$ settings (we did not evaluate an aggression setting of 9 as the write performance was very poor). For the data write, we observe that the performance is acceptable for $\text{aggression} \in [1, 5]$ and decreases significantly for $\text{aggression} > 5$.

For the selection test cases, the performance results are consistent with the results from the previous tests. On portal, the performance of the z-slice selection improves significantly when using compression, while the spectra and subcube selection show comparable performance with and without compression (see Figure \textit{Selection performance results using varying gzip aggression settings (portal)}). For hopper, we again observe a general strong decrease in the selection performance when enabling compression (while, although much slower, the overall selection performance is still acceptable) (see Figures \textit{Selection performance results using varying gzip aggression settings (hopper using /project)} and \textit{Selection performance results using varying gzip aggression settings (hopper using /scratch)}

Overall, we find that for the data layouts tested here, increasing the aggression parameter above 1 does not have a large impact on the compression ratio and selection performance, whereas high aggression values may significantly decrease the data write performance. In the context of MSI data, low aggression values of $\text{aggression} \in [1, 4]$ may, therefore, be preferable when using gzip compression.

- \texttt{numpy.var(data[:, :, zstart : (zstart + 20001)])} with \texttt{zstart} being selected randomly ($\texttt{zstart} \in [0, 79999]$)

### 9.7 Local Scalability: Multi-processing

Next we tested the local scalability using Python’s multiprocessing capabilities (i.e., on a single hopper login node and portal, and not across multiple compute nodes). We again use a $100 \times 100 \times 100,000$ test dataset (using A as donor dataset) and we regenerate the dataset for each experiment (i.e., file layout + system). We repeat each computation 50 times on the same file and report the average times and standard deviations. For this test we select 20,000 z slices (i.e, 20\% of the data) and compute the variance of the data values across the slices, i.e.:

Fig. 9.22: File size and write performance using varying gzip aggression settings

---

\textsuperscript{6} Bars shown transparently indicate that the bars exceed the maximum value shown in the plot. The real value for those bars are indicated via additional text labels.
Fig. 9.23: Selection performance results using varying gzip aggression settings (portal)

Fig. 9.24: Selection performance results using varying gzip aggression settings (hopper using /project)
We parallelize the computation across processes by dividing data along the x axis, i.e., each process computes for its portion of the data:

- \( \text{numpy.var(data[xstart : xend, :, zstart : (zstart + 20001)])} \)

while \( xstart \) and \( xend \) are determined based on the process id. The result is then stored in a shared array. The average performance (with error bars indicating the standard deviation) for performing this calculation on hopper using project are shown in Figure Scaling results for computing the variance for 20000 z-slices (hopper using /project).

We compute the speed-up by using the time required using 1 process as reference (i.e., the time from the current experiment so that the speed-up using 1 processor is always equal to 1)(see Figure Speed-up results for computing the variance for 20000 z-slices (hopper using /project)). We repeated the same tests also on portal (again using the /project file systems). The results for portal are shown in Figures: i) Scaling results for computing the variance for 20000 z-slices (portal) and ii) Speed-up results for computing the variance for 20000 z-slices (portal).

The main bottleneck in this calculation is again the data selection/load. We observe that we can achieve good speed-up for larger numbers of processes using hopper compared to portal. This behavior is likely due to the better I/O (network) performance of hopper compared to portal. On portal we observe a stable speed-up for up to 5 processes (when using compression) and we achieve a \( \approx 3 \times \) speed-up. Using hopper we observe a linear speed-up for up to 8 processes. Afterwards, we still observe speed-up, however, at a lower and less stable rate. As before, we observe that using compression yields better performance on portal while reducing the performance when using hopper.

In the context of the OpenMSI web-based data viewer we typically need to extract smaller subsets of the data. We, therefore, next repeated the variance computation for 25 consecutive z slices (similar to the z slice selection use-case used in the previous sections). The timings and speed-up results on portal using default setup —i.e., with \( 4 \times 4 \times 2048 \) chunking and gzip compression, aggression=4 — are shown in Figure Timings and speed-up results for computing the variance for 25 z-slices (portal). We observe that even-though the amount of data retrieved is with 0.5MB comparably small —note the data being loaded to fulfill the query is on the order of 39.0625MB—, we can still achieve good speed-ups until 4-7 processes, afterwards the performance degrades again.
Fig. 9.26: Scaling results for computing the variance for 20000 z-slices (hopper using /project)

Fig. 9.27: Speed-up results for computing the variance for 20000 z-slices (hopper using /project)
Fig. 9.28: Scaling results for computing the variance for 20000 z-slices (portal)

Fig. 9.29: Speed-up results for computing the variance for 20000 z-slices (portal)
9.8 Discussion

9.8.1 Data Layout

Use of chunking has many benefits but choosing the correct chunk size can be complicated. In the context of MSI data, the choice of a good data layout is complicated by: i) the large difference between the size of the spatial $x/y$ and the $z$ ($m/z$, mass) dimension and ii) the need for regularly performing orthogonal data selection operations (namely selection of spectra $[x,y,:]$ and full slices in $z$ $[:, :, z]$). Our experiments have shown that a chunking of $4 \times 4 \times 2048$ may work well for most cases, i.e., it provides a good compromise in performance between the different selection operations while maintaining acceptable write performance and limiting the worst-case overhead in file size.

Our experiments have also shown that MSI data lends itself well to compression (we have seen reductions in file size on the order of $3 \times$ to $9 \times$ on real MSI datasets). We have also seen that the use of data compression (here gzip) may also improve the selection performance by reducing the amount of data that needs to be loaded from disk (see results from portal). This is particularly true for I/O (and or network) bound systems, such as portal. However, we have also seen (on hopper) that the use of compression can also decrease the selection performance due to the time required for decompression (see results hopper). Having direct access to uncompressed data may, hence, be advantageous for any analyses that require a large number of random accesses to the data from a high-performance compute systems (such as hopper) where the time required for decompression is larger than the time saved for data transfer. However, the reduced storage requirements and overall more stable selection performance results suggest that the use of compression may be advantageous.

Based on the results from this study we currently use a data layout of $4 \times 4 \times 2048$ with gzip compression and aggression=4 as default for storing MSI data using the OMSI HDF5 data format.

9.8.2 System Performance

In a serial setting we observe that hopper's /scratch filesystem generally provides better selective read performance than the /project file system. This is not unexpected and overall the performance of both file systems appears to be sufficient to perform most common MSI analysis tasks. However, based on the test results it appears that
on portal: i) the selection performance is highly dependent on system load (which appeared to be high on many occasions) and ii) the selection performance is bound by the performance of the I/O system. Since the performance using /project is significantly higher on hopper login nodes than on portal, it appears that the bandwidth of the network connection of portal to the GPFS /project file system may be too low to ensure a reliably fast operation of web-applications that require repeated access to large datasets. This, however, is essential for production use of science gateways that aim to make advanced data viewing and analysis capabilities accessible to the application community (such as the OpenMSI science gateway for MSI data). While we were able to achieve usable I/O performance on portal through the use of chunking and compression, we have also seen that even from /project we can achieve significantly better performance on hopper when storing the data in a raw, uncompressed form (i.e., we should be able to get much better selective I/O performance on portal using /project). While portal provides a good platform for development of science gateways and web applications, the question remains whether portal is powerful enough—with respect to both compute, I/O and network performance—to provide a reliable, high-performance platform for hosting production science gateways.

9.8.3 Future Work

In future we plan to evaluate the performance of the different data layouts in a: i) local parallel (multi-processing) and ii) distribution parallel (MPI) environment.

While gzip is one of the most widely available compression algorithms in HDF5 it is not necessarily one with the best performance. In future we plan to evaluate the use of other compression algorithms—such as, szip ⁷, LZF ⁸, HDF5 N-Bit Filter ⁹, HDF5 Scale+Offset Filter ¹⁰ etc.—to further improve the overall read performance and to reduce the larger overheads for compression we have seen on hopper. In addition to compression, HDF5 also provides a shuffle filter ¹¹ which can potentially further improve the effectiveness of data compression ¹².

---

⁷ http://www.hdfgroup.org/doc_resource/SZIP/
⁸ http://h5py.alfven.org/lzf/
⁹ http://www.hdfgroup.org/HDF5/doc/RM/RM_H5P.html#Property-SetNbit
¹⁰ http://www.hdfgroup.org/HDF5/doc/RM/RM_H5P.html#Property-SetScaleoffset
10.1 Subpackages

The omsi package defines the main BASTet software stack used, e.g., by the OpenMSI project. It contains a large range of functionality for interacting analyses, OpenMSI HDF5 data files, analysis workflows and other related infrastructure. The following is a rough overview of the various packages and modules:

- **Analysis** [omsi.analysis] Package containing the base classes that facilitate the integration of new analysis with the BASTet software stack (e.g., the file format) and collection of specific analysis functionality.

- **Data Format** [omsi.dataformat] Package for implementation and specification of file formats. In particular this package contains the base API for interacting with OpenMSI HDF5 datasets.

- **Workflow** [omsi.workflow] Package with modules for specification and execution of analysis tasks and complex analysis workflows.

- **Data Structures** [omsi.datastructure] Package with a collection of various data structures and related classes used throughout the software stack, e.g., for metadata, analysis parameter data, runtime information data etc.

- **Shared** [omsi.shared] Package used to implement shared functionality and helper functions.
• **Tools** *omsi.tools* Package for collecting tools (e.g., command-line programs) to help with particular tasks. This includes, e.g., tools for data conversion, document generation, etc.

  - *omsi.tools* Package for collecting tools (e.g., command-line programs)
  - *omsi.tools.misc* Collection of miscellaneous tools.

• **Templates** *omsi.templates* This package provides a collection of code templates to ease the development of additional components, e.g., analysis modules. As such, this package is NOT intended for direct usage but is rather just a library of code templates.

  - *omsi.templates* This package provides a collection of code templates to ease the development of additional components, e.g., analysis modules.

• **Examples** *omsi.examples* Package with a collection of various misc. example scripts.

  - *omsi.examples* Package with a collection of various misc.

### 10.1.1 analysis Package

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>omsi.analysis</em></td>
<td>Package containing the base classes that facilitate the integration of new analysis with the BASTet software stack (e.g., the file format) and collection of specific analysis functionality.</td>
</tr>
<tr>
<td><em>omsi.analysis.base</em></td>
<td>Module specifying the base analysis API for integrating new analysis functionality.</td>
</tr>
<tr>
<td><em>omsi.analysis.generic</em></td>
<td>Generic analysis class used to represent analyses of unknown types.</td>
</tr>
<tr>
<td><em>omsi.analysis.analysis_views</em></td>
<td>Helper module with functions and classes for interfacing with different analysis algorithms.</td>
</tr>
<tr>
<td><em>omsi.analysis.compound_stats</em></td>
<td>Package containing shared third-party code modules included in the analysis package.</td>
</tr>
<tr>
<td><em>omsi.analysis.findpeaks</em></td>
<td>Package of peak-finding related analysis modules.</td>
</tr>
<tr>
<td><em>omsi.analysis.msi_filtering</em></td>
<td>Module with third-party modules, functions, classes used in the MSI filtering analysis.</td>
</tr>
<tr>
<td><em>omsi.analysis.multivariate_stats</em></td>
<td>Multivariate statistics analysis</td>
</tr>
</tbody>
</table>

**analysis Package**

Package containing the base classes that facilitate the integration of new analysis with the BASTet software stack (e.g., the file format) and collection of specific analysis functionality.

```python
class *omsi.analysis.*analysis_data*(name='undefined', data=None, dtype='float32')
```

Define an output dataset for the analysis that should be written to the *omsi* HDF5 file.

The class can be used like a dictionary but restricts the set of keys that can be used to the following required keys which should be provided during initilization.

**Required Keyword Arguments:**

- **name** – The name for the dataset in the HDF5 format
- **data** – The numpy array to be written to HDF5. The data write function *omsi_file_experiment.create_analysis* used for writing of the data to file can in principal also handle other primitive data types by explicitly converting them to numpy. However, in this case the dtype is determined based on the numpy conversion and correct behavior is not guaranteed. I.e., even single scalars should be stored as a 1D numpy array here. Default value is None which is mapped to np.empty(shape=(0), dtype=dtype) in __init__
**dtype** – The data type to be used during writing. For standard numpy data types this is just the dtype of the dataset, i.e., ['data'].dtype. Other allowed datatypes are:

- For string: omsi_format.str_type (omsi_format is located in omsi.dataformat.omsi_file)
- To generate data links: ana_hdf5link (analysis_data)

```python
ana_hdf5link = -1
```

class omsi.analysis.analysis_base

Bases: omisi.datastructures.analysis_data.parameter_manager

Base class for omsi analysis functionality. The class provides a large set of functionality designed to facilitate storage of analysis data in the omsi HDF5 file format. The class also provides a set of functions to enable easy integration of new analysis with the OpenMSI web-based viewer (see Viewer functions below for details).

**Slicing:**

This class supports basic slicing to access data stored in the main member variables. By default the data is retrieved from __data_list and the __getitem__(key) function, which implements the [] operator, returns __data_list[key]['data']. The key is a string indicating the name of the parameter to be retrieved. If the key is not found in the __data_list then the function will try to retrieve the data from self.parameters list instead. By adding “parameter/key” or “dependency/key” one may also explicitly retrieve values from the parameters.

**Instance Variables:**

- **analysis_identifier** – Define the name for the analysis used as key in search operations
- **__data_list** – List of analysis_data to be written to the HDF5 file. Derived classes need to add all data that should be saved for the analysis in the omsi HDF5 file to this dictionary. See omsi.analysis.analysis_data for details.
- **parameters** – List of parameter_data objects of all analysis parameters (including those that may have dependencies).
- **data_names** – List of strings of all names of analysis output datasets. These are the target keys for __data_list.
- **profile_time_and_usage** – Boolean indicating whether we should profile the execute_analysis(...) function when called as part of the execute(...) function. The default value is false. Use the enable_time_and_usage_profiling(...) function to determine which profiling should be performed. The time_and_usage profile uses pythons cProfile (or Profile) to monitor how often and for how long particular parts of the analysis code executed.
- **profile_memory** – Boolean indicating whether we should monitor memory usage (line-by-line) when executing the execute_analysis(...) function. The default value is false. Use the enable_time_and_usage_profiling(...) function to determine which profiling should be performed.
- **omsi_analysis_storage** – List of omsi_file_analysis object where the analysis is stored. The list may be empty.
- **mpi_comm** – In case we are running with MPI, this is the MPI communicator used for running the analysis. Default is MPI.COMM_WORLD.
- **mpi_root** – In case we are running with MPI, this is the root rank where data is collected to (e.g., runtime data and analysis results)
• **update_analysis** – If the value is True, then we should execute the analysis before using the outputs. If False, then the analysis has been executed with the current parameter settings.

• **driver** – Workflow driver to be used when executing multiple analyses, e.g., via execute_recursive or execute_all. Default value is None in which case a new default driver will be used each time we execute a workflow.

### Execution Functions:

- **execute**: Then main function the user needs to call in order to execute the analysis

- **execute_analysis**: This function needs to be implemented by child classes of *analysis_base* to implement the specifics of executing the analysis.

### I/O functions:

These functions can be optionally overwritten to control how the analysis data should be written/read from the omsi HDF5 file. Default implementations are provided here, which should be sufficient for most cases.

- **add_custom_data_to_omsi_file**: The default implementation is empty as the default data write is managed by the *omsi_file_experiment.create_analysis()* function. Overwrite this function, in case that the analysis needs to write data to the HDF5 omsi file beyond what the default omsi data API does.

- **read_from_omsi_file**: The default implementation tries to reconstruct the original data as far as possible, however, in particular in case that a custom add_custom_data_to_omsi_file function has been implemented, the default implementation may not be sufficient. The default implementation reconstructs: i) analysis_identifier and reads all custom data into ii) __data_list. Note, an error will be raised in case that the analysis type specified in the HDF5 file does not match the analysis type specified by get_analysis_type(). This function can be optionally overwritten to implement a custom data read.

### Viewer functions:

Several convenient functions are used to allow the OpenMSI online viewer to interact with the analysis and to visualize it. The default implementations provided here simply indicate that the analysis does not support the data access operations required by the online viewer. Overwrite these functions in the derived analysis classes in order to interface them with the viewer. All viewer-related functions start with **v***.

NOTE: the default implementation of the viewer functions defined in *analysis_base* are designed to take care of the common requirement for providing viewer access to data from all dependencies of an analysis. In many cases, the default implementation is often still called at the end of custom viewer functions.

NOTE: The viewer functions typically support a viewer_option parameter. viewer_option=0 is expected to refer to the analysis itself.

- **v_qslice**: Retrieve/compute data slices as requested via qslice URL requests. The corresponding view of the DJANGO data access server already translates all input parameters and takes care of generating images/plots if needed. This function is only responsible for retrieving the data.

- **v_qspectrum**: Retrieve/compute spectra as requested via qspectrum URL requests. The corresponding view of the DJANGO data access server already translates all input parameters and takes care of generating images/plots if needed. This function is only responsible for retrieving the data.

- **v_qmz**: Define the m/z axes for image slices and spectra as requested by qspectrum URL requests.

- **v_qspectrum_viewer_options**: Define a list of strings, describing the different viewer options available for the analysis for qspectrum requests (i.e., **v_qspectrum**). This feature allows the analysis developer to define multiple different visualization modes for the analysis. For example, when performing a data reduction (e.g., PCA or NMF) one may want to show the raw spectra or the loadings vector of the projection in the spectrum view (**v_qspectrum**). By providing different viewer options we allow the user to decide which option they are most interested in.
v_qslice_viewer_options: Define a list of strings, describing the different viewer options available for the analysis for qslice requests (i.e., v_qslice). This feature allows the analysis developer to define multiple different visualization modes for the analysis. For example, when performing a data reduction (e.g., PCA or NMF) one may want to show the raw spectra or the loadings vector of the projection in the spectrum view (v_qspectrum). By providing different viewer options we allow the user to decide which option they are most interested in.

Initialize the basic data members

add_custom_data_to_omsi_file(analysis_group)

This function can be optionally overwritten to implement a custom data write function for the analysis to be used by the omsi_file API.

Note, this function should be used only to add additional data to the analysis group. The data that is written by default is still written by the omsi_file_experiment.create_analysis() function, i.e., the following data is written by default: i) analysis_identifier, ii) get_analysis_type, iii) __data_list, iv) parameters, v) runinfo. Since the omsi_file_experiment.create_analysis() function takes care of setting up the basic structure of the analysis storage (including the subgroups for storing parameters and data dependencies) this setup can generally be assumed to exist before this function is called. This function is called automatically at the end omsi_file.experiment.create_analysis() (i.e., actually omsi_file_analysis.__populate_analysis__(..) so that this function typically does not need to be called explicitly.

Parameters

- analysis_group – The h5py.Group object where the analysis is stored.

add_parameter (name, help, dtype=<type 'unicode'>, required=False, default=None, choices=None, data=None, group=None)

Add a new parameter for the analysis. This function is typically used in the constructor of a derived analysis to specify the parameters of the analysis.

Parameters

- name – The name of the parameter
- help – Help string describing the parameter
- type – Optional type. Default is string.
- required – Boolean indicating whether the parameter is required (True) or optional (False). Default False.
- default – Optional default value for the parameter. Default None.
- choices – Optional list of choices with allowed data values. Default None, indicating no choices set.
- data – The data assigned to the parameter. None by default.
- group – Optional group string used to organize parameters. Default None, indicating that parameters are automatically organized by driver class (e.g. in required and optional parameters)

Raies ValueError is raised if the parameter with the given name already exists.

analysis_identifier_defined()

Check whether the analysis identifier is defined by the user, i.e., set to value different than undefined.

:returns: bool

check_ready_to_execute()

Check if all inputs are ready to determine if the analysis is ready to run.

Returns List of omsi_analysis_parameter objects that are not ready. If the returned list is empty, then the analysis is ready to run.
clear_analysis()  
Clear all analysis data—i.e., parameter, dependency data, output results, runtime data

clear_analysis_data()  
Clear the list of analysis data

clear_and_restore(analysis_manager=None, resave=False)  
Clear all analysis data and restore the results from file

Parameters

• analysis_manager – Instance of omsi_analysis_manager (e.g., an omsi_file_experiment) where the analysis should be saved.

• resave – Boolean indicating whether the analysis should be saved again, even if it has been saved before. This parameter only has effect if analysis_manager is given.

Returns self, i.e., the updated analysis object with all data replaced with HDF5 references

clear_parameter_data()  
Clear the list of parameter data

clear_run_info_data()  
Clear the runtime information data

define_missing_parameters()  
Called by the execute function before self.update_analysis_parameters to set any required parameters that have not been defined to their respective default values.

This function may be overwritten in child classes to customize the definition of default parameter values and to apply any modifications (or checks) of parameters before the analysis is executed. Any changes applied here will be recorded in the parameter of the analysis.

enable_memory_profiling(enable=True)  
Enable or disable line-by-line profiling of memory usage of execute_analysis.

Parameters enable_memory (bool) – Enable (True) or disable (False) line-by-line profiling of memory usage

Raises ImportError is raised if a required package for profiling is not available.

enable_time_and_usage_profiling(enable=True)  
Enable or disable profiling of time and usage of code parts of execute_analysis.

Parameters enable (bool) – Enable (True) or disable (False) profiling

Raises ImportError is raised if a required package for profiling is not available.

execute(**kwargs)  
Use this function to run the analysis.

Parameters kwargs – Parameters to be used for the analysis. Parameters may also be set using the __setitem__ mechanism or as batches using the set_parameter_values function.

Returns This function returns the output of the execute analysis function.

Raises AnalysisReadyError in case that the analysis is not ready to be executed. This may be the case, e.g., when a dependent input parameter is not ready to be used.

classmethod execute_all(force_update=False, executor=None)  
Execute all analysis instances that are currently defined.

Parameters
• **force_update** – Boolean indicating whether we should force that all analyses are executed again, even if they have already been run with the same settings before. False by default.

• **executor** – Optional workflow executor to be used for the execution of all analyses. The executor will be cleared and then all analyses will be added to executor. Default value is None, in which case the function creates a default executor to be used.

  **Returns** The workflow executor used

  **execute_analysis()**
  Implement this function to implement the execution of the actual analysis.

  This function may not require any input parameters. All input parameters are recorded in the parameters and dependencies lists and should be retrieved from there, e.g., using basic slicing self[paramName]

  Input parameters may be added for internal use ONLY. E.g., we may add parameters that are used internally to help with parallelization of the execute_analysis function. Such parameters are not recorded and must be strictly optional so that analysis_base.execute(...) can call the function.

  **Returns** This function may return any developer-defined data. Note, all output that should be recorded must be put into the data list.

  **execute_recursive(**kwargs)**
  Recursively execute this analysis and all its dependencies if necessary

  We use a workflow driver to control the execution. To define the workflow driver we can set the self.driver variable. If no workflow driver is given (i.e., self.driver==None), then the default driver will be created. To change the default driver, see omsi.workflow.base.workflow_executor_base.DEFAULT_EXECUTOR_CLASS

  **Parameters** **kwargs** – Parameters to be used for the analysis. Parameters may also be set using the __setitem__ mechanism or as batches using the set_parameter_values function.

  **Returns** Same as execute

  **get_all_analysis_data()**
  Get the complete list of all analysis datasets to be written to the HDF5 file

  **get_all_dependency_data()**
  Get the complete list of all direct dependencies to be written to the HDF5 file

  **Returns** List of parameter_data objects that define dependencies.

  **get_all_dependency_data_recursive()**
  Get the complete list of all direct dependencies to be written to the HDF5 file

  **Returns** List of parameter_data objects that define dependencies.

  **get_all_parameter_data(exclude_dependencies=False)**
  Get the complete list of all parameter datasets to be written to the HDF5 file

  **Parameters** **exclude_dependencies** – Boolean indicating whether we should exclude parameters that define dependencies from the list

  **get_all_run_info()**
  Get the dict with the complete info about the last run of the analysis

  **get_analysis_data(index)**
  Given the index return the associated dataset to be written to the HDF5 file

  :param index: Retrun the index entry of the private member __data_list.

  **get_analysis_data_by_name(dataname)**
  Given the key name of the data return the associated analysis_data object.
Parameters **dataname** – Name of the analysis data requested from the private __data_list member.

**Returns** The analysis_data object or None if not found.

**get_analysis_data_names()**
Get a list of all analysis dataset names.

**get_analysis_identifier()**
Return the name of the analysis used as key when searching for a particular analysis

**classmethod get_analysis_instances()**
Generator function used to iterate through all instances of analysis_base. The function creates references for all weak references stored in cls._analysis_instances and returns the references if it exists and cleans up the any invalid references after the iteration is complete. :return: References to analysis_base objects

**get_analysis_type()**
Return a string indicating the type of analysis performed

**static get_default_dtypes()**
Get a list of available default dtypes used for analyses. Same as data_dtypes.get_dtypes().

**static get_default_parameter_groups()**
Get a list of commonly used parameter groups and associated descriptions.
Use of default groups provides consistency and allows other system to design custom behavior around the semantic of parameter groups

**Returns** Dictionary where the keys are the short names of the groups and the values are dicts with following keys:value pairs: ‘name’ , ‘description’. Use the ‘name’ to define the group to be used.

**get_help_string()**
Get a string describing the analysis.

**Returns** Help string describing the analysis and its parameters

**get_memory_profile_info()**
Based on the memory profile of the execute_analysis(..) function get the string describing the line-by-line memory usage.

**Returns** String describing the memory usage profile. None is returned in case that no memory profiling data is available.

**get_num_analysis_data()**
Return the number of analysis datasets to be written to the HDF5 file

**get_num_dependency_data()**
Return the number of dependencies to be written to the HDF5 file

**get_num_parameter_data()**
Return the number of parameter datasets to be written to the HDF5 file

**get_omsi_analysis_storage()**
Get a list of known locations where this analysis has been saved.

**Returns** List of omsi.dataformat.omsi_file.analysis. omsi_file_analysis objects where the analysis is saved.

**get_parameter_data(index)**
Given the index return the associated dataset to be written to the HDF5 file

:param index : Return the index entry of the private member parameters.
get_parameter_data_by_name (dataname)
  Given the key name of the data return the associated parameter_data object.

  Parameters dataname – Name of the parameter requested from the parameters member.

  Returns The parameter_data object or None if not found

get_parameter_names ()
  Get a list of all parameter dataset names (including those that may define dependencies.

get_profile_stats_object (consolidate=True, stream=None)
  Based on the execution profile of the execute_analysis(..) function get pstats.Stats object to help
  with the interpretation of the data.

  Parameters
   • consolidate – Boolean flag indicating whether multiple stats (e.g., from multiple
     cores) should be consolidated into a single stats object. Default is True.
   • stream – The optional stream parameter to be used fo the pstats.Stats object.

  Returns A single pstats.Stats object if consolidate is True. Otherwise the function returns a
  list of pstats.Stats objects, one per recorded statistic. None is returned in case that the stats
  objects cannot be created or no profiling data is available.

has_omsi_analysis_storage ()
  Check whether a storage location is known where the anlaysis has been saved.

  Returns Boolean indicating whether self.omsi_analysis_storage is not empty

keys ()
  Get a list of all valid keys, i.e., a combination of all input parameter and output names.

  Returns List of strings with all input parameter and output names.

classmethod locate_analysis (data_object, include_parameters=False)
  Given a data_object try to locate the analysis that creates the object as an output of its execution (and
  optionally analyses that have the object as an input).

  Parameters
   • data_object – The data object of interest.
   • include_parameters – Boolean indicating whether also input parameters should be
     considered in the search in addition to the outputs of an analysis

  Returns dependency_dict pointing to the relevant object or None in case the object was not
  found.

read_from_omsi_file (analysis_object, load_data=True, load_parameters=True,
  load_runtime_data=True, dependencies_omsi_format=True, ignore_type_conflict=False)
  This function can be optionally overwritten to implement a custom data read.

  The default implementation tries to reconstruct the original data as far as possible, however, in particular
  in case that a custom add_custom_data_to_omsi_file function has been implemented, the default imple-
  mentation may not be sufficient. The default implementation reconstructs: i) analysis_identifier and reads
  all custom data into iii)__data_list. Note, an error will be raised in case that the analysis type specified in
  the HDF5 file does not match the analysis type specified by get_analysis_type()

  Parameters
   • analysis_object – The omsi_file_analysis object associated with the hdf5 data group
     with the analysis data_list
- **load_data** – Should the analysis data be loaded from file (default) or just stored as h5py data objects.

- **load_parameters** – Should parameters be loaded from file (default) or just stored as h5py data objects.

- **load_runtime_data** – Should runtime data be loaded from file (default) or just stored as h5py data objects.

- **dependencies_omsi_format** – Should dependencies be loaded as omsi_file API objects (default) or just as h5py objects.

- **ignore_type_conflict** – Set to True to allow the analysis to be loaded into the current analysis object even if the type indicated in the file does not match the class. Default value is False. This behavior can be useful when different analysis have compatible data structures or when we want to load the data into a generic analysis container, e.g., analysis_generic.

**Returns** `bool` Boolean indicating whether the data was read successfully

**Raise** `TypeError` A type error will be raised in case that the analysis type specified by the file does not match the analysis type provided by `self.get_analysis_type()`

**record_execute_analysis_outputs** *(analysis_output)*  
Function used internally by `execute` to record the output of the custom `execute_analysis(...) function` to the `_data_list`.  
This function may be overwritten in child classes in order to customize the behavior for recording data outputs. Eg., for some analyses one may only want to record a particular set of outputs, rather than all outputs generated by the analysis.

**Parameters** **analysis_output** – The output of the `execute_analysis(...) function` to be recorded

**results_ready()**  
Check whether the results of the analysis are ready to be used :return: `Boolean`

**set_analysis_identifier** *(identifier)*  
Set the name of the analysis to `identifier`  
Side Effects: This function modifies `self.analysis_identifier`

**Parameters** **identifier** *(str)* – The new analysis identifier string to be used (should be unique)

**set_parameter_values** *(**kwargs)*  
Set all parameters given as input to the function. The inputs are placed in the `self.parameters` list. If the parameter refers to an existing h5py.Dataset, h5py.Group, managed h5py object, or is an instance of an existing omis_analysis_base object, then a dependency_dict will be created and stored as value instead.

**Parameters** **kwargs**  
- `**kwargs` Dictionary of keyword arguments. All keys are expected to be strings. All values are expected to be either i) numpy arrays, ii) int, float, str or unicode variables, iii) h5py.Dataset or h5py.Group, iv) or any the omsi_file API class objects. For iii) and iv) one may provide a tuple consisting of the dataobject t[0] and an additional selection string t[1].

**update_analysis_parameters** *(**kwargs)*  
Record the analysis parameters passed to the `execute()` function.

The default implementation simply calls the `set_parameter_values(...) function`. This function may be overwritten to customize the behavior of how parameters are recorded by the `execute` function.

**Parameters** **kwargs** – Dictionary of keyword arguments with the parameters passed to the `execute(…) function`
classmethod v_qmz

Parameters

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed
- `qslice_viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
- `qspectrum_viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

Returns

The following four arrays are returned by the analysis:

- `mzSpectra` : Array with the static mz values for the spectra.
- `labelSpectra` : Label for the spectral mz axis
- `mzSlice` : Array of the static mz values for the slices or None if identical to the mzSpectra.
- `labelSlice` : Label for the slice mz axis or None if identical to labelSpectra.
- `values_x`: The values for the x axis of the image (or None)
- `label_x`: Label for the x axis of the image
- `values_y`: The values for the y axis of the image (or None)
- `label_y`: Label for the y axis of the image
- `values_z`: The values for the z axis of the image (or None)
- `label_z`: Label for the z axis of the image

classmethod v_qslice

Parameters

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed
- `z` – Selection string indicating which z values should be selected.
- `viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:,zmin:zmax].

Raises

NotImplementedError in case that v_qslice is not supported by the analysis.

classmethod v_qslice_viewer_options

Parameters

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed.

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

Parameters

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually
a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

classmethod `v_qspectrump (analysis_object, x, y, viewer_option=0)`

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

Developer Note: h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

Parameters

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed
- `x` – x selection string
- `y` – y selection string
- `viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array.

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

classmethod `v_qspectrump_viewer_options (analysis_object)`

Get a list of strings describing the different default viewer options for the analysis for qspectrump. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrump should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

Parameters `analysis_object` – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrump requests (i.e., v_qspectrump(...) is not available).

`write_analysis_data (analysis_group=None)`

This function is used to write the actual analysis data to file. If not implemented, then the omsi_file_analysis API’s default behavior is used instead.
**Parameters** **analysis_group** – The h5py.Group object where the analysis is stored. May be None on cores that do not perform any writing but which need to participate in communication, e.g., to collect data for writing.

**exception** **omsi.analysis.AnalysisReadyError** *(value, params=None)*

Bases: exceptions.Exception

Custom exception used to indicate that an analysis is not ready to execute.

Initialize the AnalysisReadyError

**Parameters**

- **value** – Error message string
- **params** – Optional list of dependent parameters that are not ready to be used.

**class** **omsi.analysis.analysis_generic** *(name_key='undefined')*

Bases: omsi.analysis.base.analysis_base

This analysis class is used if the specific analysis type is unknown, e.g., when loading custom user-defined analysis data that may have not be available in the standard omsi package used.

Initialize the basic data members

**Parameters** **name_key** – The name for the analysis

**DEFAULT_OUTPUT_PREFIX** = `'output_'`

**execute** (**kwargs**)

Overwrite the default implementation of execute to update parameter specifications/types when wrapping functions where the types are not known a priori.

**Parameters** **kwargs** – Custom analysis parameters

**Returns** The result of execute_analysis()

**execute_analysis** ()

Nothing to do here.

**classmethod** **from_function** *(analysis_function, output_names=None, parameter_specs=None, name_key='undefined')*

Create a generic analysis class for a given analysis function.

This functionality is useful to ease quick scripting on analyses but should not be used in production.

**NOTE:** __analysis_function is a reserved parameter name used to store the analysis function and may not be used as an input parameter for the analysis function.

**Parameters**

- **analysis_function** – The analysis function to be wrapped for provenance tracking and storage
- **output_names** – Optionally, define a list of the names of the outputs
- **parameter_specs** – Optional list of omsi.datastructures.analysis_data.parameter_data with additional information about the parameters of the function.
- **name_key** – The name for the analysis, i.e., the analysis identifier

**Returns** A new generic analysis class

**classmethod** **get_analysis_type** ()

Return a string indicating the type of analysis performed
get_real_analysis_type()
This class is designed to handle generic (including unknown) types of analysis. In cases, e.g., were this
class is used to store analysis data from an HDF5 file we may have an actual analysis type available even
if we do not have a special analysis class may not be available in the current installation

read_from_omsi_file(analysis_object, load_data=True, load_parameters=True,
load_runtime_data=True, dependencies_omsi_format=True, ignore_type_conflict=False)
See omsi.analysis.analysis_base.read_from_omsi_file(...) for details. The function is overwritten here
mainly to initialize the self.real_analysis_type instance variable but otherwise uses the default behavior.

classmethod v_qmz(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
Implement support for qmz URL requests for the viewer

classmethod v_qslice(analysis_object, z, viewer_option=0)
Implement support for qslice URL requests for the viewer

classmethod v_qslice_viewer_options(analysis_object)
Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)
Implement support for qspectrum URL requests for the viewer

classmethod v_qspectrum_viewer_options(analysis_object)
Define which viewer_options are supported for qspectrum URL’s

write_analysis_data(analysis_group=None)
This function is used to write the actual analysis data to file. If not implemented, then the
omsi_file_analysis API’s default behavior is used instead.

Parameters
analysis_group -- The h5py.Group object where the analysis is stored. May
be None on cores that do not perform any writing but which need to participate in communi-
cation, e.g., to collect data for writing.

class omsi_analysis.omsi_findpeaks_global(name_key='undefined')
Bases: omsi_analysis.base.analysis_base
Basic global peak detection analysis. The default implementation computes the peaks on the average spectrum
and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.

TODO: The current version assumes 2D data

Initialize the basic data members

execute_analysis()
Execute the global peak finding for the given msidata and mzdata.

classmethod v_qmz(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
Implement support for qmz URL requests for the viewer

classmethod v_qslice(analysis_object, z, viewer_option=0)
Implement support for qslice URL requests for the viewer

classmethod v_qslice_viewer_options(analysis_object)
Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)
Implement support for qspectrum URL requests for the viewer

classmethod v_qspectrum_viewer_options(analysis_object)
Define which viewer_options are supported for qspectrum URL’s
class omsi.analysis.omsi_findpeaks_local (name_key='undefined')

Bases: omsi.analysis.base.analysis_base

Class defining a basic gloabl peak finding. The default implementation computes the peaks on the average spectrum and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.

TODO: The current version assumes 2D data

Initialize the basic data members

execute_analysis (msidata_subblock=None)

Execute the local peak finder for the given msidata.

Parameters

msidata_subblock – Optional input parameter used for parallel execution of the analysis only. If msidata_subblock is set, then the given subblock will be processed in SERIAL instead of processing self['msidata'] in PARALLEL (if available). This parameter is strictly optional and intended for internal use only.

classmethod v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)

Implement support for qmz URL requests for the viewer

classmethod v_qslicen (analysis_object, z, viewer_option=0)

Implement support for qslice URL requests for the viewer

classmethod v_qslice_viewer_options (analysis_object)

Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)

Implement support for qspectrum URL requests for the viewer

classmethod v_qspectrum_viewer_options (analysis_object)

Define which viewer_options are supported for qspectrum URL’s

write_analysis_data (analysis_group=None)

This function is used to write the actual analysis data to file. If not implemented, then the omsi_file_analysis API’s default behavior is used instead.

Parameters

analysis_group – The h5py.Group object where the analysis is stored.

class omsi.analysis.omsi_nmf (name_key='undefined')

Bases: omsi.analysis.base.analysis_base

Class defining a basic nmf analysis.

The function has primarily been tested we MSI datasets but should support arbitrary n-D arrays (n>=2). The last dimension of the input array must be the spectrum dmensions.

Initialize the basic data members

execute_analysis ()

Execute the nmf for the given msidata

classmethod v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)

Implement support for qmz URL requests for the viewer

classmethod v_qslice (analysis_object, z, viewer_option=0)

Implement support for qslice URL requests for the viewer

classmethod v_qslice_viewer_options (analysis_object)

Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)

Implement support for qspectrum URL requests for the viewer
classmethod v_qspectrum_viewer_options(analysis_object)
    Define which viewer_options are supported for qspectrum URL's

class omsi.analysis.omsi_cx(name_key='undefined')
    Bases: omsi.analysis.base.analysis_base
    Class used to implement CX factorization on MSI data.
    Initialze the basic data members

    classmethod comp_lev_exact(A, k, axis)
    This function computes the column or row leverage scores of the input matrix.
    Parameters
    • A – n-by-d matrix
    • k – rank parameter, k <= min(n,d)
    • axis – 0: compute row leverage scores; 1: compute column leverage scores.
    Returns 1D array of leverage scores. If axis = 0, the length of lev is n. otherwise, the length of lev is d.

dimension_index = {'pixelDim': 1, 'imageDim': 0}

execute_analysis()
    EDIT_ME:
    Replace this text with the appropriate documentation for the analysis. Describe what your analysis does and how a user can use it. Note, a user will call the function execute(...) which takes care of storing parameters, collecting execution data etc., so that you only need to implement your analysis, the rest is taken care of by analysis_base. omsi uses Sphynx syntax for the documentation.
    Keyword Arguments:

    Parameters mydata – ...

    classmethod v_qmz(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
    Get the mz axes for the analysis

    Parameters
    • analysis_object – The omsi_file_analysis object for which slicing should be performed
    • qslice_viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
    • qspectrum_viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

    Returns
    The following four arrays are returned by the analysis:
    • mz_spectra : Array with the static mz values for the spectra.
    • label_spectra : Lable for the spectral mz axis
    • mz_slice : Array of the static mz values for the slices or None if identical to the mz_spectra.
    • label_slice : Lable for the slice mz axis or None if identical to label_spectra.
classmethod v_qslice (analysis_object, z, viewer_option=0)
Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

Parameters

• analysis_object – The omsi_file_analysis object for which slicing should be performed

• z – Selection string indicating which z values should be selected.

• viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:,:,zmin:zmax].

classmethod v_qslice_viewer_options (analysis_object)
Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspec should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

Parameters analysis_object – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

classmethod v_qspec (analysis_object, x, y, viewer_option=0)
Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

Developer Note: h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

Parameters

• analysis_object – The omsi_file_analysis object for which slicing should be performed

• x – x selection string

• y – y selection string

• viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array.

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed,
e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

```python
classmethod v_qspectrumanalyzer_options(analysis_object)
    Get a list of strings describing the different default viewer options for the analysis for qspectrumanalyzer. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrumanalyzer should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

    Parameters
    analysis_object -- The omis_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

    Returns
    List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrumanalyzer requests (i.e., v_qspectrumanalyzer(...) is not available).
```

```python
class omisi_analysis.omsi_kmeans(name_key='undefined')
    Bases: omisi_analysis.base.analysis_base
    Class defining a basic nmf analysis for a 2D MSI data file or slice of the data

    Initialize the basic data members

    execute_analysis()
    Execute the kmeans clustering for the given msidata
```

```python
class omisi_analysis.omsi_tic_norm(name_key='undefined')
    Bases: omisi_analysis.base.analysis_base
    TIC Normalization analysis.

    Initialize the basic data members

    execute_analysis()
    Normalize the data based on the total intensity of a spectrum or the intensities of a select set of ions.

    Calculations are performed using a memory map approach to avoid loading all data into memory. TIC normalization can as such be performed even on large files (assuming sufficient disk space).

    Keyword Arguments:

    Parameters

    • msidata (h5py.dataset or numpy array (3D)) -- The input MSI dataset

    • mzdata -- The mz axis of the dataset

    • maxCount -- ...

    • mzTol -- ...

    • infIons -- List of informative ions
```

```python
record_execute_analysis_outputs(analysis_output)
    We are not returning any outputs here, but we are going to record them manually. :param analysis_output: The output of the execute_analysis(...) function.
```

```python
classmethod v_qmz(analysis_object, qslice_viewer_option=0, qspectrumanalyzer_option=0)
    Get the mz axes for the analysis

    Parameters
• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **qslice_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.

• **qspectrum_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

Returns

The following four arrays are returned by the analysis:

• **mz_spectra** : Array with the static mz values for the spectra.

• **label_spectra** : Label for the spectral mz axis

• **mz_slice** : Array of the static mz values for the slices or None if identical to the **mz_spectra**.

• **label_slice** : Label for the slice mz axis or None if identical to **label_spectra**.

**classmethod v_qslice**(analysis_object, z, viewer_option=0)

Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

Parameters

• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **z** – Selection string indicting which z values should be selected.

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:, :, zmin:zmax].

**classmethod v_qslice_viewer_options**(analysis_object)

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

Parameters

**analysis_object** – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns

List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

**classmethod v_qspectrum**(analysis_object, x, y, viewer_option=0)

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

Developer Note: h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

Parameters
• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **x** – x selection string

• **y** – y selection string

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array/

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

```python
classmethod v_qspectrums_viewer_options(analysis_object)
```

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**param analysis_object** The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**returns** List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., v_qspectrums(...) is not available).

**base Module**

Module specifying the base analysis API for integrating new analysis with the toolkit and the OpenMSI science gateway.

**exception** omsi.analysis.base.AnalysisReadyError(value, params=None)

**Bases:** exceptions.Exception

Custom exception used to indicate that an analysis is not ready to execute.

Initialize the AnalysisReadyError

**Parameters**

• **value** – Error message string

• **params** – Optional list of dependent parameters that are not ready to be used.

**class** omsi.analysis.base.analysis_base

**Bases:** omsi.datastructures.analysis_data.parameter_manager
Base class for omsi analysis functionality. The class provides a large set of functionality designed to facilitate storage of analysis data in the omsi HDF5 file format. The class also provides a set of functions to enable easy integration of new analysis with the OpenMSI web-based viewer (see Viewer functions below for details).

Slicing:

This class supports basic slicing to access data stored in the main member variables. By default the data is retrieved from __data_list and the __getitem__(key) function, which implements the [...] operator, returns __data_list[key][‘data’]. The key is a string indicating the name of the parameter to be retrieved. If the key is not found in the __data_list then the function will try to retrieve the data from self.parameters list instead. By adding “parameter/key” or “dependency/key” one may also explicitly retrieve values from the parameters.

Instance Variables:

Variables

• **analysis_identifier** – Define the name for the analysis used as key in search operations
• **__data_list** – List of analysis_data to be written to the HDF5 file. Derived classes need to add all data that should be saved for the analysis in the omsi HDF5 file to this dictionary. See omsi.analysis.analysis_data for details.
• **parameters** – List of parameter_data objects of all analysis parameters (including those that may have dependencies).
• **data_names** – List of strings of all names of analysis output datasets. These are the target keys for __data_list.
• **profile_time_and_usage** – Boolean indicating whether we should profile the execute_analysis(...) function when called as part of the execute(...) function. The default value is false. Use the enable_time_and_usage_profiling(…) function to determine which profiling should be performed. The time_and_usage profile uses python’s cProfile (or Profile) to monitor how often and for how long particular parts of the analysis code executed.
• **profile_memory** – Boolean indicating whether we should monitor memory usage (line-by-line) when executing the execute_analysis(...) function. The default value is false. Use the enable_time_and_usage_profiling(…) function to determine which profiling should be performed.
• **omsi_analysis_storage** – List of omsi_file_analysis object where the analysis is stored. The list may be empty.
• **mpi_comm** – In case we are running with MPI, this is the MPI communicator used for running the analysis. Default is MPI.Comm_world/
• **mpi_root** – In case we are running with MPI, this is the root rank where data is collected to (e.g., runtime data and analysis results)
• **update_analysis** – If the value is True, then we should execute the analysis before using the outputs. If False, then the analysis has been executed with the current parameter settings.
• **driver** – Workflow driver to be used when executing multiple analyses, e.g., via execute_recursive or execute_all. Default value is None in which case a new default driver will be used each time we execute a workflow.

Execution Functions:

• **execute**: Then main function the user needs to call in order to execute the analysis
• **execute_analysis**: This function needs to be implemented by child classes of analysis_base to implement the specifics of executing the analysis.
I/O functions:
These functions can be optionally overwritten to control how the analysis data should be written/read from the omsi HDF5 file. Default implementations are provided here, which should be sufficient for most cases.

- **add_custom_data_to_omsi_file**: The default implementation is empty as the default data write is managed by the `omsi_file_experiment.create_analysis()` function. Overwrite this function, in case that the analysis needs to write data to the HDF5 omsi file beyond what the default omsi data API does.

- **read_from_omsi_file**: The default implementation tries to reconstruct the original data as far as possible, however, in particular in case that a custom add_custom_data_to_omsi_file function has been implemented, the default implementation may not be sufficient. The default implementation reconstructs: i) analysis_identifier and reads all custom data into ii) __data_list. Note, an error will be raised in case that the analysis type specified in the HDF5 file does not match the analysis type specified by get_analysis_type(). This function can be optionally overwritten to implement a custom data read.

Viewer functions:
Several convenient functions are used to allow the OpenMSI online viewer to interact with the analysis and to visualize it. The default implementations provided here simply indicate that the analysis does not support the data access operations required by the online viewer. Overwrite these functions in the derived analysis classes in order to interface them with the viewer. All viewer-related functions start with `v__...`

NOTE: the default implementation of the viewer functions defined in `analysis_base` are designed to take care of the common requirement for providing viewer access to data from all dependencies of an analysis. In many cases, the default implementation is often sill called at the end of custom viewer functions.

NOTE: The viewer functions typically support a viewer_option parameter. viewer_option=0 is expected to refer to the analysis itself.

- **v_qslice**: Retrieve/compute data slices as requested via qslice URL requests. The corresponding view of the DJANGO data access server already translates all input parameters and takes care of generating images/plots if needed. This function is only responsible for retrieving the data.

- **v_qspectrum**: Retrieve/compute spectra as requested via qspectrum URL requests. The corresponding view of the DJANGO data access server already translates all input parameters and takes care of generating images/plots if needed. This function is only responsible for retrieving the data.

- **v_qmz**: Define the m/z axes for image slices and spectra as requested by qspectrum URL requests.

- **v_qspectrum_viewer_options**: Define a list of strings, describing the different viewer options available for the analysis for qspectrum requests (i.e., v_qspectrum). This feature allows the analysis developer to define multiple different visualization modes for the analysis. For example, when performing a data reduction (e.g., PCA or NMF) one may want to show the raw spectra or the loadings vector of the projection in the spectrum view (v_qspectrum). By providing different viewer options we allow the user to decide which option they are most interested in.

- **v_qslice_viewer_options**: Define a list of strings, describing the different viewer options available for the analysis for qslice requests (i.e., v_qslice). This feature allows the analysis developer to define multiple different visualization modes for the analysis. For example, when performing a data reduction (e.g., PCA or NMF) one may want to show the raw spectra or the loadings vector of the projection in the spectrum view (v_qspectrum). By providing different viewer options we allow the user to decide which option they are most interested in.

Initialize the basic data members

**add_custom_data_to_omsi_file** *(analysis_group)*
This function can be optionally overwritten to implement a custom data write function for the analysis to be used by the omsi_file API.
Note, this function should be used only to add additional data to the analysis group. The data that is written by default is still written by the `omsi_file_experiment.create_analysis()` function, i.e., the following data is written by default: i) analysis_identifier , ii) get_analysis_type, iii) __data_list, iv) parameters, v) runinfo . Since the `omsi_file_experiment.create_analysis()` functions takes care of setting up the basic structure of the analysis storage (included the subgroubs for storing parameters and data dependencies) this setup can generally be assumed to exist before this function is called. This function is called automatically at the end `omsi_file_experiment.create_analysis()` (i.e, actually `omsi_file_analysis.__populate_analysis__(..) `so that this function typically does not need to be called explicitly.

Parameters **analysis_group** – The h5py.Group object where the analysis is stored.

**add_parameter** (name, help, dtype=<type ‘unicode’>, required=False, default=None, choices=None, data=None, group=None)

Add a new parameter for the analysis. This function is typically used in the constructor of a derived analysis to specify the parameters of the analysis.

Parameters

- **name** – The name of the parameter
- **help** – Help string describing the parameter
- **type** – Optional type. Default is string.
- **required** – Boolean indicating whether the parameter is required (True) or optional (False). Default False.
- **default** – Optional default value for the parameter. Default None.
- **choices** – Optional list of choices with allowed data values. Default None, indicating no choices set.
- **data** – The data assigned to the parameter. None by default.
- **group** – Optional group string used to organize parameters. Default None, indicating that parameters are automatically organized by driver class (e.g. in required and optional parameters)

**Raises** ValueError is raised if the parameter with the given name already exists.

**analysis_identifier_defined**()

Check whether the analysis identifier is defined by the user, i.e., set to value different than undefined

:returns: bool

**check_ready_to_execute**()

Check if all inputs are ready to determine if the analysis is ready to run.

**Returns** List of omsi_analysis_parameter objects that are not ready. If the returned list is empty, then the analysis is ready to run.

**clear_analysis**()

Clear all analysis data—i.e., parameter, dependency data, output results, runtime data

**clear_analysis_data**()

Clear the list of analysis data

**clear_and_restore** (analysis_manager=None, resave=False)

Clear all analysis data and restore the results from file

Parameters

- **analysis_manager** – Instance of omsi_analysis_manager (e.g., an omsi_file_experiment) where the analysis should be saved.
• *resave* – Boolean indicating whether the analysis should be saved again, even if it has been saved before. This parameter only has effect if *analysis_manager* is given.

**Returns** self, i.e., the updated analysis object with all data replaced with HDF5 references

clear_parameter_data()  
Clear the list of parameter data

clear_run_info_data()  
Clear the runtime information data

define_missing_parameters()  
Called by the execute function before self.update_analysis_parameters to set any required parameters that have not been defined to their respective default values.

This function may be overwritten in child classes to customize the definition of default parameter values and to apply any modifications (or checks) of parameters before the analysis is executed. Any changes applied here will be recorded in the parameter of the analysis.

enable_memory_profiling(*enable=True*)  
Enable or disable line-by-line profiling of memory usage of execute_analysis.

**Parameters**  
*enable* (bool) – Enable (True) or disable (False) line-by-line profiling of memory usage

**Raises** ImportError is raised if a required package for profiling is not available.

enable_time_and_usage_profiling(*enable=True*)  
Enable or disable profiling of time and usage of code parts of execute_analysis.

**Parameters**  
*enable* (bool) – Enable (True) or disable (False) profiling

**Raises** ImportError is raised if a required package for profiling is not available.

execute(**kwargs**)  
Use this function to run the analysis.

**Parameters**  
**kwargs** – Parameters to be used for the analysis. Parameters may also be set using the __setitem__ mechanism or as batches using the set_parameter_values function.

**Returns** This function returns the output of the execute analysis function.

**Raises** AnalysisReadyError in case that the analysis is not ready to be executed. This may be the case, e.g., when a dependent input parameter is not ready to be used.

classmethod execute_all(*force_update=False, executor=None*)  
Execute all analysis instances that are currently defined.

**Parameters**  
• *force_update* – Boolean indicating whether we should force that all analyses are executed again, even if they have already been run with the same settings before. False by default.

• *executor* – Optional workflow executor to be used for the execution of all analyses. The executor will be cleared and then all analyses will be added to executor. Default value is None, in which case the function creates a default executor to be used.

**Returns** The workflow executor used

eexecute_analysis()  
Implement this function to implement the execution of the actual analysis.

This function may not require any input parameters. All input parameters are recorded in the parameters and dependencies lists and should be retrieved from there, e.g., using basic slicing self[paramName]
Input parameters may be added for internal use ONLY. E.g., we may add parameters that are used internally to help with parallelization of the execute_analysis function. Such parameters are not recorded and must be strictly optional so that analysis_base.execute(...) can call the function.

**Returns** This function may return any developer-defined data. Note, all output that should be recorded must be put into the data list.

**execute_recursive** (**kwargs**)  
Recursively execute this analysis and all its dependencies if necessary

We use a workflow driver to control the execution. To define the workflow driver we can set the self.driver variable. If no workflow driver is given (i.e., self.driver==None), then the default driver will be created. To change the default driver, see omsi.workflow.base.workflow_executor_base.DEFAULT_EXECUTOR_CLASS

**Parameters** **kwargs** – Parameters to be used for the analysis. Parameters may also be set using the __setitem__ mechanism or as batches using the set_parameter_values function.

**Returns** Same as execute

**get_all_analysis_data** ()  
Get the complete list of all analysis datasets to be written to the HDF5 file

**get_all_dependency_data** ()  
Get the complete list of all direct dependencies to be written to the HDF5 file

NOTE: These are only the direct dependencies as specified by the analysis itself. Use get_all_dependency_data_recursive(...) to also get the indirect dependencies of the analysis due to dependencies of the dependencies themselves.

**Returns** List of parameter_data objects that define dependencies.

**get_all_parameter_data** (exclude_dependencies=False)  
Get the complete list of all parameter datasets to be written to the HDF5 file

**Parameters** exclude_dependencies – Boolean indicating whether we should exclude parameters that define dependencies from the list

**get_all_run_info** ()  
Get the dict with the complete info about the last run of the analysis

**get_analysis_data** (**index**)  
Given the index return the associated dataset to be written to the HDF5 file

:param index : Retrun the index entry of the private member __data_list.

**get_analysis_data_by_name** (**dataname**)  
Given the key name of the data return the associated analysis_data object.

**Parameters** dataname – Name of the analysis data requested from the private __data_list member.

**Returns** The analysis_data object or None if not found.

**get_analysis_data_names** ()  
Get a list of all analysis dataset names.

**get_analysis_identifier** ()  
Return the name of the analysis used as key when searching for a particular analysis

**classmethod** **get_analysis_instances** ()  
Generator function used to iterate through all instances of analysis_base. The function creates references for all weak references stored in cls._analysis_instances and returns the references if it exists and cleans up any invalid references after the iteration is complete. :return: References to analysis_base objects.
get_analysis_type()
Return a string indicating the type of analysis performed

static get_default_dtypes()
Get a list of available default dtypes used for analyses. Same as data_dtypes.get_dtypes().

static get_default_parameter_groups()
Get a list of commonly used parameter groups and associated descriptions.
Use of default groups provides consistency and allows other system to design custom behavior around the
semantic of parameter groups

    Returns: Dictionary where the keys are the short names of the groups and the values are dicts
    with following keys:value pairs: ‘name’, ‘description’. Use the ‘name’ to define the group
to be used.

get_help_string()
Get a string describing the analysis.

    Returns: Help string describing the analysis and its parameters

get_memory_profile_info()
Based on the memory profile of the execute_analysis(..) function get the string describing the line-by-line
memory usage.

    Returns: String describing the memory usage profile. None is returned in case that no memory
profiling data is available.

get_num_analysis_data()
Retrun the number of analysis datasets to be written to the HDF5 file

get_num_dependency_data()
Return the number of dependencies to be written to the HDF5 file

get_num_parameter_data()
Return the number of parameter datasets to be written to the HDF5 file

get_omsi_analysis_storage()
Get a list of known locations where this analysis has been saved.

    Returns: List of omsi.dataformat.omsi_file_analysis. omsi_file_analysis objects where the anal-
ysis is saved.

get_parameter_data(index)
Given the index return the associated dataset to be written to the HDF5 file

    :param index: Return the index entry of the private member parameters.

get_parameter_data_by_name(dataname)
Given the key name of the data return the associated parameter_data object.

    Parameters dataname – Name of the parameter requested from the parameters member.

    Returns: The parameter_data object or None if not found

get_parameter_names()
Get a list of all parameter dataset names (including those that may define dependencies.

get_profile_stats_object(consolidate=True, stream=None)
Based on the execution profile of the execute_analysis(..) function get pstats.Stats object to help
with the interpretation of the data.

    Parameters
• **consolidate** – Boolean flag indicating whether multiple stats (e.g., from multiple cores) should be consolidated into a single stats object. Default is True.

• **stream** – The optional stream parameter to be used for the pstats.Stats object.

**Returns** A single pstats.Stats object if consolidate is True. Otherwise the function returns a list of pstats.Stats objects, one per recorded statistic. None is returned in case that the stats objects cannot be created or no profiling data is available.

### has_omsi_analysis_storage()

Check whether a storage location is known where the analysis has been saved.

**Returns** Boolean indicating whether self.omsi_analysis_storage is not empty

### keys()

Get a list of all valid keys, i.e., a combination of all input parameter and output names.

**Returns** List of strings with all input parameter and output names.

### classmethod locate_analysis(data_object, include_parameters=False)

Given a data_object try to locate the analysis that creates the object as an output of its execution (and optionally analyses that have the object as an input).

**Parameters**

• **data_object** – The data object of interest.

• **include_parameters** – Boolean indicating whether also input parameters should be considered in the search in addition to the outputs of an analysis

**Returns** dependency_dict pointing to the relevant object or None in case the object was not found.

### read_from_omsi_file(analysis_object, load_data=True, load_parameters=True, load_runtime_data=True, dependencies_omsi_format=True, ignore_type_conflict=False)

This function can be optionally overwritten to implement a custom data read.

The default implementation tries to reconstruct the original data as far as possible, however, in particular in case that a custom add_custom_data_to_omsi_file function has been implemented, the default implementation may not be sufficient. The default implementation reconstructs: i) analysis_identifier and reads all custom data into iii) __data_list. Note, an error will be raised in case that the analysis type specified in the HDF5 file does not match the analysis type specified by get_analysis_type()
data structures or when we want to load the data in to a generic analysis container, e.g.,
analysis_generic.

Returns bool  Boolean indicating whether the data was read successfully

Raise TypeError : A type error will be raised in case that the analysis type specified by the file
does not match the analysis type provided by self.get_analysis_type()

record_execute_analysis_outputs (analysis_output)
Function used internally by execute to record the output of the custom execute_analysis(...) function to the
__data_list.
This function may be overwritten in child classes in order to customize the behavior for recording data
outputs. Eg., for some analyses one may only want to record a particular set of outputs, rather than all
outputs generated by the analysis.

Parameters analysis_output – The output of the execute_analysis(...) function to be
recorded

results_ready ()
Check whether the results of the analysis are ready to be used :return: Boolean

set_analysis_identifier (identifier)
Set the name of the analysis to identifier
Side Effects: This function modifies self.analysis_identifier

Parameters identifier (str) – The new analysis identifier string to be used (should be
unique)

set_parameter_values (**kwargs)
Set all parameters given as input to the function. The inputs are placed in the self.parameters list. If the
parameter refers to an existing h5py.Dataset, h5py.Group, managed h5py object, or is an instance of an
existing omis_analysis_base object, then a dependency_dict will be created and stored as value instead.

Parameters kwargs – Dictionary of keyword arguments. All keys are expected to be strings.
All values are expected to be either i) numpy arrays, ii) int, float, str or unicode variables, iii)
h5py.Dataset or h5py.Group, iv) or any the omsi_file API class objects. For iii) and iv) one
may provide a tuple consisting of the dataobject t[0] and an additional selection string t[1].

update_analysis_parameters (**kwargs)
Record the analysis parameters passed to the execute() function.
The default implementation simply calls the set_parameter_values(...) function. This function may be
overwritten to customize the behavior of how parameters are recorded by the execute function.

Parameters kwargs – Dictionary of keyword arguments with the parameters passed to the
execute(...) function

classmethod v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
Get the mz axes for the analysis

Parameters

• analysis_object – The omsi_file_analysis object for which slicing should be performed

• qslice_viewer_option – If multiple default viewer behaviors are available for a
given analysis then this option is used to switch between them for the qslice URL pattern.

• qspectrum_viewer_option – If multiple default viewer behaviors are available for a
given analysis then this option is used to switch between them for the qspectrum URL pattern.
Returns

The following four arrays are returned by the analysis:

- **mzSpectra**: Array with the static mz values for the spectra.
- **labelSpectra**: Label for the spectral mz axis
- **mzSlice**: Array of the static mz values for the slices or None if identical to the mzSpectra.
- **labelSlice**: Label for the slice mz axis or None if identical to labelSpectra.
- **values_x**: The values for the x axis of the image (or None)
- **label_x**: Label for the x axis of the image
- **values_y**: The values for the y axis of the image (or None)
- **label_y**: Label for the y axis of the image
- **values_z**: The values for the z axis of the image (or None)
- **label_z**: Label for the z axis of the image

**classmethod v_qslice(analysise_object, z, viewer_option=0)**

Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

**Parameters**

- **analysise_object**: The omsi_file_analysis object for which slicing should be performed
- **z**: Selection string indicating which z values should be selected.
- **viewer_option**: If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:]zmin:zmax:.

**Raises**

NotImplementedError in case that v_qslice is not supported by the analysis.

**classmethod v_qslice_viewer_options(analysise_object)**

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**Parameters**

- **analysise_object**: The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**Returns**

List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

**classmethod v_qspectrum(analysise_object, x, y, viewer_option=0)**

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

**Developer Note:** h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

**Parameters**
• **analysis_object** – The omsi_file_analysis object for which slicing should be performed
• **x** – x selection string
• **y** – y selection string
• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array.
2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

**classmethod v_qspectrum_viewer_options (analysis_object)**

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**Parameters**

**analysis_object** – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**Returns** List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., v_qspectrum(...) is not available).

**write_analysis_data (analysis_group=None)**

This function is used to write the actual analysis data to file. If not implemented, then the omsi_file_analysis API’s default behavior is used instead.

**Parameters**

**analysis_group** – The h5py.Group object where the analysis is stored. May be None on cores that do not perform any writing but which need to participate in communication, e.g., to collect data for writing.

**generic Module**

Generic analysis class used to represent analyses of unknown type, e.g., when loading a custom user-defined analysis from file for which the indicate class may not be available with the local installation. In this case we want to at least be able to load and investigate the data.

**class** omsi.analysis.generic.analysis_generic (name_key='undefined')

**Bases:** omsi.analysis.base.analysis_base

This analysis class is used if the specific anlaysis type is unknown, e.g., when loading custom user-defined analysis data that may have not be available in the standard omsi package used.
Initialize the basic data members

Parameters name_key – The name for the analysis

DEFAULT_OUTPUT_PREFIX = 'output_

execute(**kwargs)

Overwrite the default implementation of execute to update parameter specifications/types when wrapping functions where the types are not known a priori.

Parameters kwargs – Custom analysis parameters

Returns The result of execute_analysis()

execute_analysis()

Nothing to do here.

classmethod from_function(analysis_function, output_names=None, parameter_specs=None, name_key='undefined')

Create a generic analysis class for a given analysis function.

This functionality is useful to ease quick scripting on analyses but should not be used in production.

NOTE: __analysis_function is a reserved parameter name used to store the analysis function and may not be used as an input parameter for the analysis function.

Parameters

• analysis_function – The analysis function to be wrapped for provenance tracking and storage

• output_names – Optionally, define a list of the names of the outputs

• parameter_specs – Optional list of omsi.datastructures.analysis_data.parameter_data with additional information about the parameters of the function.

• name_key – The name for the analysis, i.e., the analysis identifier

Returns A new generic analysis class

classmethod get_analysis_type()

Return a string indicating the type of analysis performed

classmethod get_real_analysis_type()

This class is designed to handle generic (including unkown) types of analysis. In cases, e.g., were this class is used to store analysis data from an HDF5 file we may have an actual analysis type available even if we do not have a special analysis class may not be available in the current installation

read_from_omsi_file(analysis_object, load_data=True, load_parameters=True, load_runtime_data=True, dependencies_omsi_format=True, ignore_type_conflict=False)

See omsi.analysis.analysis_base.read_from_omsi_file(...) for details. The function is overwritten here mainly to initialize the self.real_analysis_type instance variable but otherwise uses the default behavior.

classmethod v_qmz(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)

Implement support for qmz URL requests for the viewer

classmethod v_qslice(analysis_object, z, viewer_option=0)

Implement support for qslice URL requests for the viewer

classmethod v_qslice_viewer_options(analysis_object)

Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)

Implement support for qspectrum URL requests for the viewer
classmethod v_qспектр Viewer Options (analysis_object)
    Define which viewer_options are supported for qспектр URL's

write_analysis_data (analysis_group=None)
    This function is used to write the actual analysis data to file. If not implemented, then the
    omsi_file_analysis API's default behavior is used instead.

    Parameters
    ----------
    analysis_group -- The h5py.Group object where the analysis is stored. May be None on cores that do not perform any writing but which need to participate in communication, e.g., to collect data for writing.

omsi.analysis.generic.bastet_analysis (output_names=None, parameter_specs=None, name_key='undefined')
    Decorator used to wrap a function and replace it with an analysis_generic object that behaves like a function but adds the ability for saving the analysis to file and tracking provenance

    This is essentially the same as analysis_generic.from_function(...).

    Parameters
    ----------
    func -- The function to be wrapped
    output_names -- Optional list of strings with the names of the outputs
    parameter_specs -- Optional list of omsi.datastructures.analysis_data.parameter_data with additional information about the parameters of the function.
    name_key -- Optional name for the analysis, i.e., the analysis identifier

    Returns
    -------
    analysis_generic instance for the wrapped function

analysis_views Module

Helper module with functions and classes for interfacing with different analysis algorithms. Many of these functions are used to ease interaction with the analysis module in a generic fashion, without having to explicitly know about all the different available modules, e.g., we can just look up modules by name and interact with them directly.

class omsi.analysis.analysis_views.analysis_views
    Bases: object

    Helper class for interfacing different analysis algorithms with the web-based viewer
    Nothing to do here.

classmethod analysis_name_to_class (class_name)
    Convert the given string indicating the class to a python class.

    Parameters
    ----------
    class_name -- Name of the analysis class. This may be a fully qualified name, e.g., omsi.analysis.multivariate_stat.omsi_nmf or a name relative to the omsi.analysis module, e.g., multivariate_stat.omsi_nmf.

    Raises
    -----
    Attribute error in case that the class cannot be restored.

classmethod available_analysis ()
    Get all available analysis, i.e., all analysis that are subclasses of analysis_base.

    Returns
    -------
    Dictionary where the dict-keys are the full qualified name of the module and the values are the analysis class corresponding to that module.

classmethod available_analysis_descriptions ()
    Get all available analysis, i.e., all analysis that are subclasses of analysis_base. For each analysis compile the list of input parameters, outputs, the corresponding class etc.
Returns Dictionary where the dict-keys are the full qualified name of the module and the values are dicts with class, list of analysis parameters names, list of analysis outputs.

classmethod get_axes (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
Get the mz axes for the analysis

Parameters
• analysis_object – The omsi_file_analysis object for which slicing should be performed
• qslice_viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
• qspectrum_viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

Returns
The following four arrays are returned by the analysis:
• mz_spectra : Array with the static mz values for the spectra.
• label_spectra : Label for the spectral mz axis
• mz_slice : Array of the static mz values for the slices or None if identical to the mz_spectra.
• label_slice : Lable for the slice mz axis or None if identical to label_spectra.
• values_x: The values for the x axis of the image (or None)
• label_x: Label for the x axis of the image
• values_y: The values for the y axis of the image (or None)
• label_y: Label for the y axis of the image
• values_z: The values for the z axis of the image (or None)
• label_z: Label for the z axis of the image

classmethod get_qslice_viewer_options (analysis_object)
Get a list of strings describing the different default viewer options for qslice.

Parameters analysis_object – The omsi_file_analysis object for which slicing should be performed.

Returns Array of strings indicating the different available viewer options. The array may be empty if now viewer_options are available, i.e., get_slice and get_spectrum are undefined for the given analysis.

classmethod get_qspectrum_viewer_options (analysis_object)
Get a list of strings describing the different default viewer options for qspectrum.

Parameters analysis_object – The omsi_file_analysis object for which slicing should be performed.

Returns Array of strings indicating the different available viewer options. The array may be empty if now viewer_options are available, i.e., get_slice and get_spectrum are undefined for the given analysis.

classmethod get_slice (analysis_object, z, operations=None, viewer_option=0)
Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer
Parameters

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed
- **z** – Selection string indicating which z values should be selected.
- **operations** – JSON string with list of dictionaries or a python list of dictionaries. Each dict specifies a single data transformation or data reduction that are applied in order. See omsi.shared.omsi_data_selection.transform_and_reduce_data(...) for details.
- **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:,zmin:zmax].

```
@classmethod
get_spectra(analysis_object, x, y, operations=None, viewer_option=0)
```

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

**Developer Note:** h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

Parameters

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed
- **x** – x selection string
- **y** – y selection string
- **operations** – JSON string with list of dictionaries or a python list of dictionaries. Each dict specifies a single data transformation or data reduction that are applied in order. See omsi.shared.omsi_data_selection.transform_and_reduce_data(...) for details.
- **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

2D or 3D numpy array of the requested spectra. The mass (m/z) axis must be the last axis.

```
@classmethod
supports_slice(analysis_object)
```

Get whether a default slice selection behavior is defined for the analysis.

Parameters **analysis_object** – The omsi_file_analysis object for which slicing should be performed

Returns Boolean indicating whether get_slice(...) is defined for the analysis object.

```
@classmethod
supports_spectra(analysis_object)
```

Get whether a default spectra selection behavior is defined for the analysis.

Parameters **analysis_object** – The omsi_file_analysis object for which slicing should be performed.

Returns Boolean indicating whether get_spectra(...) is defined for the analysis object.

Subpackages

**findpeaks Package**
**findpeaks Package**  Package of peak-finding related analysis modules.

```python
class omsi.analysis.findpeaks.omsi_findpeaks_global(name_key=’undefined’)
    Bases: omsi.analysis.base.analysis_base
    Basic global peak detection analysis. The default implementation computes the peaks on the average spectrum and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.
    TODO: The current version assumes 2D data
    Initialize the basic data members
    execute_analysis()
        Execute the global peak finding for the given msidata and mzdata.
    classmethod v_qmz(analysis_object, qslice_viewer_option=0, qslice_viewer_option=0)
        Implement support for qmz URL requests for the viewer
    classmethod v_qslice(analysis_object, z, viewer_option=0)
        Implement support for qslice URL requests for the viewer
    classmethod v_qslice_viewer_options(analysis_object)
        Define which viewer_options are supported for qspectrum URL’s
    classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)
        Implement support for qspectrum URL requests for the viewer
    classmethod v_qspectrum_viewer_options(analysis_object)
        Define which viewer_options are supported for qspectrum URL’s

class omsi.analysis.findpeaks.omsi_findpeaks_local(name_key=’undefined’)
    Bases: omsi.analysis.base.analysis_base
    Class defining a basic global peak finding. The default implementation computes the peaks on the average spectrum and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.
    TODO: The current version assumes 2D data
    Initialize the basic data members
    execute_analysis(msidata_subblock=None)
        Execute the local peak finder for the given msidata.
            Parameters
            msidata_subblock – Optional input parameter used for parallel execution of the analysis only. If msidata_subblock is set, then the given subblock will be processed in SERIAL instead of processing self[‘msidata’] in PARALLEL (if available). This parameter is strictly optional and intended for internal use only.
    classmethod v_qmz(analysis_object, qslice_viewer_option=0, qslice_viewer_option=0)
        Implement support for qmz URL requests for the viewer
    classmethod v_qslice(analysis_object, z, viewer_option=0)
        Implement support for qslice URL requests for the viewer
    classmethod v_qslice_viewer_options(analysis_object)
        Define which viewer_options are supported for qspectrum URL’s
    classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)
        Implement support for qspectrum URL requests for the viewer
    classmethod v_qspectrum_viewer_options(analysis_object)
        Define which viewer_options are supported for qspectrum URL’s
```

10.1. Subpackages
**BASTet: Berkeley Analysis and Storage Toolkit, Release devel**

**write_analysis_data** *(analysis_group=None)*
This function is used to write the actual analysis data to file. If not implemented, then the omsi_file_analysis API's default behavior is used instead.

**Parameters**

- **analysis_group** – The h5py.Group object where the analysis is stored.

**omsi_findpeaks_global Module**
Global peak finder computing peaks and associated ion-images for the full MSI data.

**class** *omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global*(name_key='undefined')
**Bases:** *omsi.analysis.base.analysis_base*

Basic global peak detection analysis. The default implementation computes the peaks on the average spectrum and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.

TODO: The current version assumes 2D data

Initialize the basic data members

**execute_analysis** ()
Execute the global peak finding for the given msidata and mzdata.

**classmethod** *v_qmz*(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
Implement support for qmz URL requests for the viewer

**classmethod** *v_qslice*(analysis_object, z, viewer_option=0)
Implement support for qslice URL requests for the viewer

**classmethod** *v_qslice_viewer_options*(analysis_object)
Define which viewer_options are supported for qspectrum URL's

**classmethod** *v_qspectrum*(analysis_object, x, y, viewer_option=0)
Implement support for qspectrum URL requests for the viewer

**classmethod** *v_qspectrum_viewer_options*(analysis_object)
Define which viewer_options are supported for qspectrum URL's

**omsi_findpeaks_local Module**
Local peak finding analysis module.

**class** *omsi.analysis.findpeaks.omsi_findpeaks_local.omsi_findpeaks_local*(name_key='undefined')
**Bases:** *omsi.analysis.base.analysis_base*

Class defining a basic gloabl peak finding. The default implementation computes the peaks on the average spectrum and then computes the peak-cube data, i.e., the values for the detected peaks at each pixel.

TODO: The current version assumes 2D data

Initialize the basic data members

**execute_analysis** *(msidata_subblock=None)*
Execute the local peak finder for the given msidata.

**Parameters**

- **msidata_subblock** – Optional input parameter used for parallel execution of the analysis only. If msidata_subblock is set, then the given subblock will be processed in SERIAL instead of processing self['msidata'] in PARALLEL (if available). This parameter is strictly optional and intended for internal use only.

**classmethod** *v_qmz*(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
Implement support for qmz URL requests for the viewer

**classmethod** *v_qslice*(analysis_object, z, viewer_option=0)
Implement support for qslice URL requests for the viewer
classmethod v_qslice_viewer_options (analysis_object)
    Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)
    Implement support for qspectrum URL requests for the viewer

classmethod v_qspectrum_viewer_options (analysis_object)
    Define which viewer_options are supported for qspectrum URL’s

write_analysis_data (analysis_group=None)
    This function is used to write the actual analysis data to file. If not implemented, then the
    omsi_file_analysis API’s default behavior is used instead.

    Parameters
    "analysis_group" -- The h5py.Group object where the analysis is stored.

findpeaks.third_party Package

findpeaks.third_party Package  Package containing shared third-party code modules included here to re-
    duce the need for external dependencies when only small parts of external code are used.

findpeaks Module

class omsi.analysis.findpeaks.third_party.findpeaks.findpeaks (x, y, sizes-
    smooth, slwindow, peakheight)

    Name = ‘findpeaks’
    display()
    peakdet()
    Converted from MATLAB script at http://billauer.co.il/peakdet.html

    Currently returns two lists of tuples, but maybe arrays would be better

    function [maxtab, mintab]=peakdet(v, delta, x) %PEAKDET Detect peaks in a vector % [MAXTAB,
    MINTAB] = PEAKDET(V, DELTA) finds the local % maxima and minima (“peaks”) in the vector V.
    % MAXTAB and MINTAB consists of two columns. Column 1 % contains indices in V, and column 2 the
    found values. % % With [MAXTAB, MINTAB] = PEAKDET(V, DELTA, X) the indices % in MAXTAB
    and MINTAB are replaced with the corresponding % X-values. % % A point is considered a maximum
    peak if it has the maximal % value, and was preceded (to the left) by a value lower by % DELTA.
    % Eli Billauer, 3.4.05 (Explicitly not copyrighted). % This function is released to the public domain; Any
    use is allowed.
    sliding_window_minimum()
    A iterator which takes the size of the window, k, and an iterable, li. Then returns an iterator such that the
    ith element yielded is equal to min(list(li)[max(i - k + 1, 0):i+1]).

    Each yield takes amortized O(1) time, and overall the generator takes O(k) space.
    smoothListGaussian()  

findpeaks.experimental Package

findpeaks.experimental Package  Module with experimental analysis code, i.e., code that is not (yet) used
    in production but is under development. Often this is code that is used in a specific research.
**omsi_peakcube Module**

`omsi.analysis.findpeaks.experimental.omsi_peakcube.main` *(argv=None)*

**class` omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube`**(name_key='undefined')

Bases: `omsi.analysis.base.analysis_base`

Initialize the basic data members

```python
execute_analysis()
```

`getGlobalMz(peaksBins, peaksMZdata, HCpeaksLabels, HCLabelsList)`

`getPeakCube(peaksIntensities, peaksArrayIndex, HCpeaksLabels, HCLabelsList)`

`record_execute_analysis_outputs(analysis_output)`

We are recording our outputs manually as part of the execute function

**classmethod` v_qmz`**(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)

Implement support for qmz URL requests for the viewer

**classmethod` v_qslice`**(analysis_object, z, viewer_option=0)

Implement support for qslice URL requests for the viewer

**classmethod` v_qslice_viewer_options`**(analysis_object)

Define which viewer_options are supported for qslice URL's

**classmethod` v_qslice`**(analysis_object, x, y, viewer_option=0)

Implement support for qspectrum URL requests for the viewer

**classmethod` v_qslice_viewer_options`**(analysis_object)

Define which viewer_options are supported for qspectrum URL's

`omsi.analysis.findpeaks.experimental.omsi_peakcube.stop()`

**pfrun Module**

`omsi.analysis.findpeaks.experimental.pfrun.generateScript(scriptfile, PF-content=None, repo=None)`

`omsi.analysis.findpeaks.experimental.pfrun.main` *(argv=None)*

`omsi.analysis.findpeaks.experimental.pfrun.printHelp(thisfilename)`

`omsi.analysis.findpeaks.experimental.pfrun.queuePCjob(pcstring, therepo=None)`

`omsi.analysis.findpeaks.experimental.pfrun.run_lpf(omsiInFile, expIndex, dataIndex, ph, slw, smw)`

`omsi.analysis.findpeaks.experimental.pfrun.run_npg(omsiInFile, expIndex, dataIndex, LPFIndex, mzth, tcut)`

`omsi.analysis.findpeaks.experimental.pfrun.run_peakcube(omsiInFile, dataIndex, LPFIndex, NPGIndex)`

`omsi.analysis.findpeaks.experimental.pfrun.stop()`

**omsi_lpf Module**

`omsi.analysis.findpeaks.experimental.omsi_lpf.cl_peakfind(self, msidt, smoothsize, slwindow, peakheight)`

`omsi.analysis.findpeaks.experimental.omsi_lpf.execute_analysis(self)`

`omsi.analysis.findpeaks.experimental.omsi_lpf.main` *(argv=None)*

Then main function
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

```python
class omsi.analysis.findpeaks.experimental.omsi_lpf.omsi_lpf(name_key='undefined')
    Bases: omsi.analysis.base.analysis_base

omsi.analysis.findpeaks.experimental.omsi_lpf.v_qmz
    classmethod(function) -> method
    Convert a function to be a class method.
    A class method receives the class as implicit first argument, just like an instance method receives the instance.
    To declare a class method, use this idiom:
    
    class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)
    
    It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
    
    Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

omsi.analysis.findpeaks.experimental.omsi_lpf.v_qslice
    classmethod(function) -> method
    Convert a function to be a class method.
    A class method receives the class as implicit first argument, just like an instance method receives the instance.
    To declare a class method, use this idiom:
    
    class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)
    
    It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
    
    Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

omsi.analysis.findpeaks.experimental.omsi_lpf.v_qslice_viewer_options
    classmethod(function) -> method
    Convert a function to be a class method.
    A class method receives the class as implicit first argument, just like an instance method receives the instance.
    To declare a class method, use this idiom:
    
    class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)
    
    It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
    
    Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

omsi.analysis.findpeaks.experimental.omsi_lpf.v_qspectrum
    classmethod(function) -> method
    Convert a function to be a class method.
    A class method receives the class as implicit first argument, just like an instance method receives the instance.
    To declare a class method, use this idiom:
    
    class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)
    
    It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
    
    Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.
```

10.1. Subpackages
Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

```python
class C:
    def f(cls, arg1, arg2, ...): ...
    f = classmethod(f)
```

It can be called either on the class (e.g. `C.f()`) or on an instance (e.g. `C().f()`). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the `staticmethod` builtin.

### omsi_npg Module

```python
class omsi.analysis.findpeaks.experimental.omsi_npg.Node (label)
oclass omsi.analysis.findpeaks.experimental.omsi_npg.main (argv=None)
```

```
class omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg (name_key='undefined')
    Bases: omsi.analysis.base.analysis_base
    Initialize the basic data members
    Find (x)
    MakeSet (x)
    Union (x, y)
    execute_analysis ()
    getClustersInfo (GpeaksLabels, GLabelsList)
    getCoordIdxB (xCoord, yCoord)
    getCoordInfoB (xCoord, yCoord, peaksLabels)
    getCoordPeaksB (xCoord, yCoord)
    getNearestPeakIndex (myPeaksArray, myPeak)
    getPixelMap (Nx, Ny)
    classmethod getnpgimage (PeaksLabels, LabelsList, peaksArrayIndex, peaksIntensities, z)
    classmethod getnpgspec (PeaksLabels, LabelsList, peaksArrayIndex, peaksIntensities, xCoord, yCoord)
    myHC (labelsMMz, TreeCut)
    record_execute_analysis_outputs (analysis_output)
        We are recording our outputs manually as part of the execute function
    splitLabelsList (LabelData, Thres, SplitMax)
    classmethod v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
        Implement support for qmz URL requests for the viewer
    classmethod v_qslice (analysis_object, z, viewer_option=0)
        Implement support for qslice URL requests for the viewer
    classmethod v_qslice_viewer_options (analysis_object)
        Define which viewer options are supported for qspectrum URL's
```
classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)  
Implement support for qspectrum URL requests for the viewer

classmethod v_qspectrum_viewer_options (analysis_object)  
Define which viewer_options are supported for qspectrum URL’s

osmi.analysis.findpeaks.experimental.osmi_npg.stop()

mypeakfinder Module
osmi.analysis.findpeaks.experimental.mypeakfinder.generateScript (scriptfile,  
PFcontent=None, repo=None)

osmi.analysis.findpeaks.experimental.mypeakfinder.getJobOutput (jobname, runtime)

osmi.analysis.findpeaks.experimental.mypeakfinder.getPFcmd (pfstring, lpfstring,  
npstring, LPFIndex, NPGIndex, Skip-NPG, SkipPeakCube, IndexFile=None)

osmi.analysis.findpeaks.experimental.mypeakfinder.main (argv)

osmi.analysis.findpeaks.experimental.mypeakfinder.monitorJob (jobid, jobname,  
runtyle='pf')

osmi.analysis.findpeaks.experimental.mypeakfinder.printHelp (thisfilename)

osmi.analysis.findpeaks.experimental.mypeakfinder.stop()

multivariate_stats Package
multivariate_stats Package  Multivariate statistics analysis

class osmi.analysis.multivariate_stats.osmi_nmf (name_key='undefined')  
Bases: osmi.analysis.base.analysis_base  
Class defining a basic nmf analysis.

The function has primarily been tested we MSI datasets but should support arbitrary n-D arrays (n>=2). The last dimension of the input array must be the spectrum dimensions.

Initialize the basic data members

execute_analysis ()  
Execute the nmf for the given msidata

classmethod v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)  
Implement support for qmz URL requests for the viewer

classmethod v_qslice (analysis_object, z, viewer_option=0)  
Implement support for qslice URL requests for the viewer

classmethod v_qslice_viewer_options (analysis_object)  
Define which viewer_options are supported for qspectrum URL’s

classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)  
Implement support for qspectrum URL requests for the viewer

classmethod v_qspectrum_viewer_options (analysis_object)  
Define which viewer_options are supported for qspectrum URL’s
**Class** `omsia.analysis.multivariate_stats.omsi_cx(name_key='undefined')`

**Bases:** `omsia.analysis.base.analysis_base`

Class used to implement CX factorization on MSI data.

Initialize the basic data members

```python
classmethod comp_lev_exact(A, k, axis)
```

This function computes the column or row leverage scores of the input matrix.

**Parameters**

- `A` – n-by-d matrix
- `k` – rank parameter, k <= min(n,d)
- `axis` – 0: compute row leverage scores; 1: compute column leverage scores.

**Returns**

1D array of leverage scores. If axis = 0, the length of lev is n. otherwise, the length of lev is d.

```python
dimension_index = {'pixelDim': 1, 'imageDim': 0}
```

```python
execute_analysis()
```

EDIT_ME:

Replace this text with the appropriate documentation for the analysis. Describe what your analysis does and how a user can use it. Note, a user will call the function execute(...) which takes care of storing parameters, collecting execution data etc., so that you only need to implement your analysis, the rest is taken care of by analysis_base. omsi uses Sphynx syntax for the documentation.

**Keyword Arguments:**

- **Parameters** `mydata` – ...

```python
classmethod v_qmz(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)
```

Get the mz axes for the analysis

**Parameters**

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed
- `qslice_viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
- `qspectrum_viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

**Returns**

The following four arrays are returned by the analysis:

- `mz_spectra` : Array with the static mz values for the spectra.
- `label_spectra` : Lable for the spectral mz axis
- `mz_slice` : Array of the static mz values for the slices or None if identical to the `mz_spectra`.
- `label_slice` : Lable for the slice mz axis or None if identical to `label_spectra`.

```python
classmethod v_qslice(analysis_object, z, viewer_option=0)
```

Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

**Parameters**

- **Parameters** `analysis_object` – ...
- `z` –...
- `viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **z** – Selection string indicating which z values should be selected.

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:,:zmin:zmax].

**classmethod v_qslice_viewer_options (analysis_object)**

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**Parameters**

**analysis_object** – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**Returns**

List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

**classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)**

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

**Developer Note:** h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

**Parameters**

• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **x** – x selection string

• **y** – y selection string

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array.

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

**classmethod v_qspectrum_viewer_options (analysis_object)**

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The
default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

Parameters

**analysis_object** – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns

List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., v_qspectrum(...) is not available).

class omsi_analysis_multivariate_stats_omsi_kmeans(name_key='undefined')

**Bases:** omsi_analysis_base_analysis_base

Class defining a basic nmf analysis for a 2D MSI data file or slice of the data

Initialize the basic data members

**execute_analysis**()

Execute the kmeans clustering for the given msidata

class omsi_analysis_multivariate_stats_omsi_nmf(name_key='undefined')

**Bases:** omsi_analysis_base_analysis_base

Class defining a basic nmf analysis.

The function has primarily been tested with MSI datasets but should support arbitrary n-D arrays (n>=2). The last dimension of the input array must be the spectrum dimensions.

Initialize the basic data members

**execute_analysis**()

Execute the nmf for the given msidata

**classmethod v_qmz**(analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)

Implement support for qmz URL requests for the viewer

**classmethod v_qslice**(analysis_object, z, viewer_option=0)

Implement support for qslice URL requests for the viewer

**classmethod v_qslice_viewer_options**(analysis_object)

Define which viewer_options are supported for qspectrum URL’s

**classmethod v_qspectrum**(analysis_object, x, y, viewer_option=0)

Implement support for qspectrum URL requests for the viewer

**classmethod v_qspectrum_viewer_options**(analysis_object)

Define which viewer_options are supported for qspectrum URL’s

**multivariate_stats.third_party Package**

**multivariate_stats.third_party Package** Package containing shared third-party code modules included here to reduce the need for external dependencies when only small parts of external code are used.
**nfm Module**  
Copyright (c) 2005-2008 Chih-Jen Lin All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. 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.
3. Neither name of copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

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 THE REGENTS OR CONTRIBUTORS 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.

```python
omsi.analysis.multivariate_stats.third_party.nmf.nlssubprob(V, W, Hinit, tol, maxiter)
```

- `H, grad`: output solution and gradient
- `iter`: #iterations used
- `V, W`: constant matrices
- `Hinit`: initial solution
- `tol`: stopping tolerance
- `maxiter`: limit of iterations

```python
omsi.analysis.multivariate_stats.third_party.nmf.nmf(V, Winit, Hinit, tol, timelimit, maxiter)
```

- `(W,H) = nmf(V,Winit,Hinit,tol,timelimit,maxiter)`: output solution
- `Winit,Hinit`: initial solution
- `tol`: tolerance for a relative stopping condition
- `timelimit, maxiter`: limit of time and iterations

**multivariate_stats.experimental Package**

**multivariate_stats.experimental Package**  
Module with experimental analysis code, i.e., code that is not (yet) used in production but is under development. Often this is code that is used in a specific research.

**msi_filtering Package**

**msi_filtering Package**  
Module with third-party modules, functions, classes used by some of the analysis modules in the containing package.

```python
class omsi.analysis.msi_filtering.omsi_tic_norm(name_key='undefined')
```

**Bases:** `omsi.analysis.base.analysis_base`

TIC Normalization analysis.

Initialize the basic data members

```python
execute_analysis()
```

- Normalize the data based on the total intensity of a spectrum or the intensities of a select set of ions.
- Calculations are performed using a memory map approach to avoid loading all data into memory. TIC normalization can as such be performed even on large files (assuming sufficient disk space).

**Keyword Arguments:**

**Parameters:**

**10.1. Subpackages**
• **msidata** (*h5py.dataset or numpu array (3D*)) – The input MSI dataset

• **mzdata** – The mz axis do the dataset

• **maxCount** – ...

• **mzTol** – ...

• **infIons** – List of informative ions

**record_execute_analysis_outputs** (*analysis_output*)

We are not returning any outputs here, but we are going to record them manually. :param analysis_output:
The output of the execute_analysis(...) function.

**classmethod v_qmz** (*analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0*)

Get the mz axes for the analysis

**Parameters**

• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **qslice_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.

• **qspectrum_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

**Returns**

The following four arrays are returned by the analysis:

• **mz_spectra** : Array with the static mz values for the spectra.

• **label_spectra** : Label for the spectral mz axis

• **mz_slice** : Array of the static mz values for the slices or None if identical to the **mz_spectra**.

• **label_slice** : Label for the slice mz axis or None if identical to **label_spectra**.

**classmethod v_qslice** (*analysis_object, z, viewer_option=0*)

Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

**Parameters**

• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **z** – Selection string indicting which z values should be selected.

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns** numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:,zmin:zmax].

**classmethod v_qslice_viewer_options** (*analysis_object*)

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the depencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.
Parameters `analysis_object` – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., `v_qslice(...)` is not available).

`classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)`
Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

Developer Note: h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

Parameters

* `analysis_object` – The omsi_file_analysis object for which slicing should be performed
* `x` – x selection string
* `y` – y selection string
* `viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array/

2. None in case that the spectra axis returned by `v_qmz` are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

`classmethod v_qspectrum_viewer_options (analysis_object)`
Get a list of strings describing the different default viewer options for the analysis for qspectrum. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

`param analysis_object` The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

`returns` List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., `v_qspectrum(...)` is not available).
omsi_tic_norm Module  Module with the TIC normalization analysis.

class omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm(name_key='undefined')
    Bases: omsi.analysis.base.analysis_base

    TIC Normalization analysis.

    Initialize the basic data members

    execute_analysis()
        Normalize the data based on the total intensity of a spectrum or the intensities of a select set of ions.
        Calculations are performed using a memory map approach to avoid loading all data into memory. TIC normalization can as such be performed even on large files (assuming sufficient disk space).

    Keyword Arguments:
        Parameters
            • msidata (h5py.dataset or numpy array (3D)) – The input MSI dataset
            • mzdata – The mz axis do the dataset
            • maxCount – ...
            • mzTol – ...
            • infIons – List of informative ions

    record_execute_analysis_outputs (analysis_output)
        We are not returning any outputs here, but we are going to record them manually. :param analysis_output:
        The output of the execute_analysis(...) function.

    classmethod v_qmz (analysis_object, qslice_viewer_option=0, qsmcetrum_viewer_option=0)
        Get the mz axes for the analysis

        Parameters
            • analysis_object – The omsi_file_analysis object for which slicing should be performed
            • qslice_viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
            • qsmcetrum_viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qsmcetrum URL pattern.

        Returns
            The following four arrays are returned by the analysis:
                • mz_spectra : Array with the static mz values for the spectra.
                • label_spectra : Lable for the spectral mz axis
                • mz_slice : Array of the static mz values for the slices or None if identical to the mz_spectra.
                • label_slice : Lable for the slice mz axis or None if identical to label_spectra.

    classmethod v_qslice (analysis_object, z, viewer_option=0)
        Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

        Parameters
• **analysis_object** – The omsi_file_analysis object for which slicing should be performed

• **z** – Selection string indicating which z values should be selected.

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns** numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:, ;, zmin:zmax].

classmethod v_qslice_viewer_options (analysis_object)

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**Parameters**

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**Returns** List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

**Developer Note:** h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

**Parameters**

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed

- **x** – x selection string

- **y** – y selection string

- **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0] & y=[1] usually a 2D array is expected. For range selections x=0:1 & y=1:2 we one usually expects a 3D array/

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

classmethod v_qspectrum_viewer_options (analysis_object)

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The
default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**param** analysis_object The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**returns** List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., v_qspectrum(...) is not available).

### msi_filtering.third_party Package

**msi_filtering.third_party Package** Package containing shared third-party code modules included here to reduce the need for external dependencies when only small parts of external code are used.

### msi_filtering.experimental Package

**msi_filtering.experimental Package** Module with experimental analysis code, i.e., code that is not (yet) used in production but is under development. Often this is code that is used in a specific research.

### omsi_filter_by_mask Module

**omsi_filter_by_mask Module** Module for performing masking for MSI data.

**class** omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask.omsi_filter_by_mask (name_key='undefined')

**Bases:** omsi.analysis.base.analysis_base

Class defining a basic mask creation a 2D MSI data file or slice of the data

Initialize the basic data members

**execute_analysis**()

**classmethod** v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)

Get the mz axes for the analysis

**Parameters**

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed

- **qslice_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.

- **qspectrum_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

**Returns**

The following four arrays are returned by the analysis:

- **mz_spectra** : Array with the static mz values for the spectra.

- **label_spectra** : Lable for the spectral mz axis

- **mz_slice** : Array of the static mz values for the slices or None if identical to the mz_spectra.
• label_slice : Lable for the slice mz axis or None if identical to label_spectra.

classmethod v_qslice (analysis_object, z, viewer_option=0)
Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

Parameters
• analysis_object – The omsi_file_analysis object for which slicing should be performed
• z – Selection string indicting which z values should be selected.
• viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:,:,zmin:zmax].

classmethod v_qslice_viewer_options (analysis_object)
Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

Parameters analysis_object – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

Returns List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

classmethod v_qspectrum (analysis_object, x, y, viewer_option=0)
Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

Developer Note: h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpary array which supports mutiple lists in the selection.

Parameters
• analysis_object – The omsi_file_analysis object for which slicing should be performed
• x – x selection string
• y – y selection string
• viewer_option – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

Returns
The following two elemnts are expected to be returned by this function :
1. 1D, 2D or 3D numpary array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array.
2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpary array with the m/z values for the spectrum (i.e., if custom

10.1. Subpackages
m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

**class** `v_qspectrum_viewer_options (analysis_object)`

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**param** `analysis_object` The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**returns** List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., v_qspectrum(...) is not available).

---

**omsi_mask_by_cluster Module**

Module for performing making a mask from cluster matrix for MSI data.

**class** `omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster` (name)

Bases: `omsi.analysis.base.analysis_base`

Class defining a basic mask creation a 2D MSI data file or slice of the data

Initialize the basic data members

**execute_analysis()**

**class** `v_qmz (analysis_object, qslice_viewer_option=0, qspectrum_viewer_option=0)`

Get the mz axes for the analysis

**Parameters**

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed
- **qslice_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
- **qspectrum_viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

**Returns**

The following four arrays are returned by the analysis:

- **mz_spectra** : Array with the static mz values for the spectra.
- **label_spectra** : Label for the spectral mz axis
- **mz_slice** : Array of the static mz values for the slices or None if identical to the mz_spectra.
- **label_slice** : Label for the slice mz axis or None if identical to label_spectra.

**class** `v_qslice (analysis_object, z, viewer_option=0)`

Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer

**Parameters**
• **analysis_object** – The omni_file_analysis object for which slicing should be performed

• **z** – Selection string indicating which z values should be selected.

• **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,zmin:zmax].

classmethod **v_qslice_viewer_options**(analysis_object)

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the spectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**Parameters**

- **analysis_object** – The omni_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**Returns**

List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

classmethod **v_qspectrum**(analysis_object, x, y, viewer_option=0)

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

**Developer Note:** h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

**Parameters**

- **analysis_object** – The omni_file_analysis object for which slicing should be performed

- **x** – x selection string

- **y** – y selection string

- **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=0\&y=1 usually a 2D array is expected. For range selections x=0:1\&y=1:2 we usually expect a 3D array.

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.

classmethod **v_qspectrum_viewer_options**(analysis_object)

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The
default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the q_spectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**param analysis_object**  The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**returns**  List of strings indicating the different available viewer options. The list should be empty if the analysis does not support q_spectrum requests (i.e., v_q_spectrum(...) is not available).

**compound_stats Package**

**compound_stats Package**  Package containing shared third-party code modules included here to reduce the need for external dependencies when only small parts of external code are used.

**omsi_score_midas Module**  MIDAS spectrum analysis

```python
class omsi.analysis.compound_stats.omsi_score_midas.omsi_score_midas(name_key='undefined')
    Bases: omsi.analysis.base.analysis_base

    Class for executing midas on an MSI or local peak finding dataset.
    Initialize the basic data members

    execute_analysis(spectrum_indexes=None, compound_list=None)
        Execute the local peak finder for the given msidata.

        Parameters

        • **spectrum_indexes** – List with a list of integer indicies of the subset of spectra that should be processed by this MPI task. If spectrum_indexes is set, then the given subblock will be processed in SERIAL instead of processing self['fpl_data'] in PARALLEL (if available). This parameter is strictly optional and intended for internal use only to facilitate the efficient parallel implementation.

        • **compound_list** – List of the compounds from the database file. This parameter is used to avoid having to read the compound database on every compute task that calls this function when running in parallel. This parameter is strictly optional and intended for internal use only to facilitate the efficient parallel implementation.

        Returns

        A tuple with an array of hit_tables with the scores for each pixel and a 2D array of pixel indices describing for each spectrum the (x,y) pixel location in the image. The hit_table is an array of (#spectra x #compounds). The hit_table is a structured numpy array with the following columns:

        • ‘score’, float, MIDAS score of row
        • ‘id’, str, database ID e.g. ‘MetaCyC_7884’
        • ‘name’, str, database name, e.g. ‘glycine’
        • ‘mass’, float, mass in Da of IDed compound
        • ‘n_peaks’, int, number of peaks in data
```
• ‘n_match’, int, number of peaks in data matched

**omsi_score_pactolus Module**  
MIDAS spectrum analysis

```python
class omsi.analysis.compound_stats.omsi_score_pactolus.omsi_score_pactolus(name_key='undefined')
Bases: omsi.analysis.base.analysis_base
```

Class for executing Pactolus on a local peak finding dataset.

Initialize the basic data members

execute_analysis(spectrum_indexes=None, file_lookup_table=None)

Execute the local peak finder for the given msidata.

**Parameters**

• **spectrum_indexes** – List with a list of integer indicies of the subset of spectra that should be processed by this MPI task. If spectrum_indexes is set, then the given subblock will be processed in SERIAL instead of processing self['fpl_data'] in PARALLEL (if available). This parameter is strictly optional and intended for internal use only to facilitate the efficient parallel implementation.

• **file_lookup_table** – The Pactolus lookup table with the list of tree files and their mass.

**Returns**

A series of numpy arrays with the score data for each pixel and a 2D array of pixel indices describing for each spectrum the (x,y) pixel location in the image.

```python
['pixel_index', 'score', 'id', 'name', 'mass', 'n_peaks', 'n_match']
```

• ‘pixel_index’, int, 2D array of pixel indices describing for each spectrum the (x,y) pixel location in the image

• ‘score’, float, MIDAS score of row

• ‘id’, str, database ID e.g. ‘MetaCyC_7884’

• ‘name’, str, database name, e.g. ‘glycine’

• ‘mass’, float, mass in Da of IDed compound

• ‘n_peaks’, int, number of peaks in data

• ‘n_match’, int, number of peaks in data matched

**compound_stats.third_party Package**

**compound_stats.third_party Package**  
Package containing shared third-party code modules included here to reduce the need for external dependencies when only small parts of external code are used.

**compound_stats.experimental Package**

**compound_stats.experimental Package**  
Module with experimental analysis code, i.e., code that is not (yet) used in production but is under development. Often this is code that is used in a specific research.
10.1.2 dataformat Package

Main module for specification of data formats. This module also contains the omsi_file module which specifies the OpenMSI HDF5 data format. In addition it defines the base class for third-party file readers (i.e., file_reader_base) and implements various basic file readers for third-party formats, e.g., img_file and mzml_file for IMG and MZML data files respectively (among others).

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omsi.dataformat.omsi_file</td>
<td>Module for specification of the OpenMSI file API.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.analysis</td>
<td>Module for managing custom analysis data in OMSI HDF5 files.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.common</td>
<td>Module for common data format classes and functionality.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.dependencies</td>
<td>Base module for managing of dependencies between data in OpenMSI files.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.experiment</td>
<td>OMSI file module for management of experiment data.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.format</td>
<td>This module defines the basic format for storing mass spectrometry imaging data.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.instrument</td>
<td>Module for managing instrument related data in OMSI files.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.main_file</td>
<td>Module for managing OpenMSI HDF5 data files.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.metadata_collection</td>
<td>Module for management of general metadata storage entities.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.methods</td>
<td>Module for management of method specific data in OMSI data.</td>
</tr>
<tr>
<td>omsi.dataformat.omsi_file.msidata</td>
<td>Module for managing MSI data in OMSI data files.</td>
</tr>
<tr>
<td>omsi.dataformat.file_reader_base</td>
<td>Module for base classes for implementation and integration of third-party file readers.</td>
</tr>
<tr>
<td>omsi.dataformat.bruckerflex_file</td>
<td>This module provides functionality for reading bruker flex mass spectrometry imaging data.</td>
</tr>
<tr>
<td>omsi.dataformat.imzml_file</td>
<td>This module provides functionality for reading imzML mass spectrometry imaging data.</td>
</tr>
<tr>
<td>omsi.dataformat.mzml_file</td>
<td>This module provides functionality for reading mzml mass spectrometry imaging data.</td>
</tr>
</tbody>
</table>

OMSI Dataformat Package

Main module for specification of the OpenMSI HDF5-based data format. The module contains various sub-modules, with the main goal to organize different categories of data.

Naming conventions for objects inside the HDF5 file are defined in the omsi_file.format module. These are used by the manager API classes to then implement the specific format.

The basic idea behind the design of the OpenMSI file format is the concept of managed objects. Managed objects are objects in an HDF5 file (usually HDF5 Groups – similar to directories just within a file) that have a corresponding interface class in the API. These classes in the API always start with the prefix omsi_file_ and inherit from omsi_file_common.omsi_file_common.

To make it easy to nest different objects, we also have the concept of manager helper classes, which encapsulate common functionality for creation and interaction with the objects when they are contained in another object. Manager helper classes follow the following naming convention omsi_<objectname>_manager, where objectname is name of the object to be managed. E.g., omsi_instrument_manager is used to help place instrument groups inside another managed object. This is done by inheriting from the given manager helper class.

This means, multiple inheritance is used in order to nest other managed modules with other interfaces. This allows us to easily encapsulate common interaction features in centralized locations and construct more complex containers simply via inheritance.

The user of multiple inheritance and super can be tricky in python. To simplify the use and ensure stability we use the following conventions:

- All omsi_file_* manager classes must except the h5py.Group object they manage as input and call super(..).__init__(managed_group) with the managed group as parameter in their __init__.
- All omsi_<objectname>_manager manager helper classes must except the h5py.Group that contains the object(s) that should be managed using the helper class as input and call super(..).__init__(managed_group) with the managed group as parameter in their in __init__.
• All `omsi_file_*` manager classes must inherit from `omsi_file.common.omsi_file_common`
• All `omsi_<objectname>_manager` manager helper classes must inherit from `object` (i.e., we use new-style classes).

**omsi_file Package**

Module for specification of the OpenMSI file API.

**format Module**

This module defines the basic format for storing mass spectrometry imaging data, metadata, and analysis in HDF5 in compliance with OpenMSI file format.

```python
class omsi.dataformat.omsi_file.format.omsi_format_analysis
    Bases: omsi.dataformat.omsi_file.format.omsi_format_common

    Specification for storing analysis related data.
    
    **Variables**
    
    • `analysis_groupname` – `analysis_`: Group with additional analysis results
    • `analysis_identifier` – `analysis_identifier`: Identifier for the analysis to enable look-up by analysis id
    • `analysis_type` – `analysis_type`: Dataset used to store the analysis type descriptor string
    • `analysis_parameter_group` – Group for storing analysis parameters. Dependent parameters are stored separately using a `omsi_format_dependencies` group.
    • `analysis_runinfo_group` – Group for storing run information, e.g., where was the analysis run, how long did it take etc.

analysis_groupname = ‘analysis_’
analysis_identifier = ‘analysis_identifier’
analysis_parameter_group = ‘parameter’
analysis_parameter_help_attr = ‘help’
analysis_runinfo_group = ‘runinfo’
analysis_type = ‘analysis_type’
current_version = ‘0.2’
```

class omsi.dataformat.omsi_file.format.omsi_format_common
    Bases: object

    Specification of common attributes, and names for the file format.
    
    **Variables**
    
    • `str_type` – `str = h5py.new_vlen(str)`: Datatype used for storing strings in hdf5
    • `type_attribute` – Name of the optional type attribute indicating which `omsi_file_*` class should be used to interact with a given group.

current_version = ‘0.1’
str_type = dtype(‘O’)
str_type_unicode = True
timestamp_attribute = ‘timestamp’
type_attribute = ‘omsi_type’
version_attribute = ‘version’

class omsi.dataformat.omsi_file.format.omsi_format_data
    Bases: omsi.dataformat.omsi_file.format.omsi_format_common

Specification for storing raw data information.

Variables

• data_groupname – The base name for the hdf5 group containing the imaging data
• dataset_name – The base name for storing raw data. In the case of MSI data, this is the
  3D data cube stored as 3D (full_cube), 2D (partial_cube) or 1D (partial_spectra) dataset,
  depending on the format_type.
• data_dependency_group – Optional group for storing data dependencies

current_version = ‘0.1’
data_groupname = ‘data_’
dataset_name = ‘data_’

class omsi.dataformat.omsi_file.format.omsi_format_dependencies
    Bases: omsi.dataformat.omsi_file.format.omsi_format_common

Specification for the management of a collection of dependencies.

Variables dependencies_groupname – dependency : Name of the group the dependencies are
  stored in.

current_version = ‘0.1’
dependencies_groupname = ‘dependency’

class omsi.dataformat.omsi_file.format.omsi_format_dependencydata
    Bases: omsi.dataformat.omsi_file.format.omsi_format_common

Specification for the storage of a single dependency.

This type of group does not have specific name to allow the user to specify a specific link_name to ease retrieval
of the data.

Variables

• dependency_parameter – parameter_name : Name of string dataset used to store the
  name of the dependent parameter
• dependency_selection – selection : Name of the string dataset used to store a selec-
  tion string if needed.
• dependency_mainname – main_name : Name of the string dataset used to store the
  description of the link to the object that this depends on.
• dependency_datasetname – ‘data_name’ : Name fo the string dataset used to store
  the name of dataset within the mainname highlevel object.

current_version = ‘0.3’
dependency_datasetname = ‘data_name’
dependency_mainname = ‘main_name’
dependency_parameter = 'parameter_name'
dependency_parameter_help_attr = 'help'
dependency_selection = 'selection'
dependency_typename = 'dependency_type'

class omsi.dataformat.omsi_file.format.omsi_format_experiment
Bases: omsi.dataformat.omsi_file.format.omsi_format_common
Specification of file format specific name conventions

Variables

- **exp_groupname** – *entry_*: The base name for a group containing data about an experiment
- **exp_identifier_name** – experiment_identifier: The identifier dataset for an experiment

```
current_version = '0.1'
exp_groupname = 'entry_'
exp_identifier_name = 'experiment_identifier'
```

class omsi.dataformat.omsi_file.format.omsi_format_file
Bases: omsi.dataformat.omsi_file.format.omsi_format_common
Specification of main-file related specific name conventions

```
current_version = '0.1'
```

class omsi.dataformat.omsi_file.format.omsi_format_instrument
Bases: omsi.dataformat.omsi_file.format.omsi_format_metadata_collection

Specification for storing instrument related information

Variables

- **instrument_groupname** – instrument: Group with information about the instrument used
- **instrument_mz_name** – mz: Name of the dataset for the instrument’s mz data values
- **instrument_name** – name: The dataset with the name of the instrument

```
current_version = '0.2'
instrument_groupname = 'instrument'
instrument_mz_name = 'mz'
instrument_name = 'name'
```

class omsi.dataformat.omsi_file.format.omsi_format_metadata_collection
Bases: omsi.dataformat.omsi_file.format.omsi_format_common
Specification of the basic format for a general-purpose metadata storage

Variables

- **metadata_collection_groupname_default** – Default name for the group where the collection of metadata is stored.
- **description_value_attribute** – The attribute to be associated with each metadata value describing the content in a human-readable form
• **unit_value_attribute** – The attribute to be associated with each metadata value describing the unit

• **ontology_value_attribute** – Optional ontology associated with a metadata value

current_version = '0.1'
description_value_attribute = 'description'
metadata_collection_groupname_default = 'metadata'
ontology_value_attribute = 'ontology'
unit_value_attribute = 'unit'

**class** omsi.dataformat.omsi_file.format.omsi_format_methods

Bases: omsi.dataformat.omsi_file.format.omsi_format_metadata_collection

Specification of the basic format for storing method-related information

**Variables**

• **methods_groupname** – methods : The group storing all the information about the method

• **methods_old_groupname** – method : The group object was refactored to methods. To ensure that old files can still be read, this variable was added and is checked as well if needed.

• **methods_name** – name : The dataset with the name of the method

current_version = '0.3'
methods_groupname = 'methods'
methods_name = 'name'
methods_old_groupname = 'sample'

**class** omsi.dataformat.omsi_file.format.omsi_format_msidata

Bases: omsi.dataformat.omsi_file.format.omsi_format_data

Specification of the basic format for storing an MSI dataset consisting of a complete 3D cube (or a 3D cube completed with 0s for missing data)

**Variables**

• **format_types** – Data layout types supported for storing MSI data.

• **mzdata_name** – Global mz axis for the MSI data cube.

• **format** – Dataset in HDF5 with the format_type descriptor.

current_version = '0.1'
format_name = 'format'
format_types = {'full_cube': 1, 'partial_cube': 2, 'partial_spectra': 3}
mzdata_name = 'mz'

**class** omsi.dataformat.omsi_file.format.omsi_format_msidata_partial_cube

Bases: omsi.dataformat.omsi_file.format.omsi_format_msidata

Specification of the basic format for storing an MSI datasets that define a partial cube with full spectra

**Variables**
• **xy_index_name** – 2D dataset indicating for each spectrum its start location in the main dataset

• **inv_xy_index_name** – 2D dataset with n rows and 2 columns indicating for each spectrum i the (x,y) pixel index the spectrum belongs to. This index is stored for convenience purposes but is not actually needed for data access.

• **shape_name** – Simple [3] indicating the true image size in x,y,mz

```
inv_xy_index_name = 'inv_xy_index'
shape_name = 'shape'
xy_index_name = 'xyindex'
```

class omsi.dataformat.omsi_file.format.omsi_format_msidata_partial_spectra
Bases: omsi.dataformat.omsi_file.format.omsi_format_msidata_partial_cube

Specification of the basic format for storing an MSI dataset of a full or partial cube with partial spectra

Variables

• **mz_index_name** – 1D dataset of the same size as the spectrum data, indicating the indices into the global m/z list

• **xy_index_end_name** – 2D dataset indicating for each spectrum its end location (index not included) in the main dataset

```
mz_index_name = 'mz_index'
xy_index_end_name = 'xyindexend'
```

class omsi.dataformat.omsi_file.common.omsi_file_common
Bases: object

Base class for definition of file format modules for the OpenMSI data format.

Use of super()

This class inherits only from object and calls super in the __init__ without parameters. In the standard design pattern of the omsi.dataformat.omsi_file module, it is, therefore, the last class we inherit from in the case of multiple inheritance.

Multiple inheritance is used in omsi.dataformat.omsi_file module when a class contains other managed objects and uses the manager classes, e.g, omsi_instrument_mangager etc. to get all the features needed to manage those objects.

All child classes of omsi_file_common also call super(..).__init__(manager_group) but using a single input parameter indicating the manager h5py.Group object that contains the given object.

Variables

• **managed_group** – The h5py.Group object managed by the class

• **name** – The path to the object in the hdf5 file. Same as managed_group.name

• **file** – The h5py.File object the managed_group is associated with. Same as managed_group.file
static create_path_string(filename, objectname)

Given the name of the file and the object path within the file, create a string describing the external reference to the data.

Parameters

• filename – The full or relative path to the file
• objectname – The object path in the HDF5 file

Returns String describing the path to the object

classmethod get_h5py_object(omsi_object, resolve_dependencies=False)

This static method is a convenience function used to retrieve the corresponding h5py interface object for any omsi file API object.

Parameters

• omni_object – omsi file API input object for which the corresponding h5py.Group, h5py.File, or h5py.Dataset object should be retrieved. The omni_object input may itself also be a h5py.Group, h5py.File, or h5py.Dataset, in which case omni_object itself is returned by the function.
• resolve_dependencies – Set to True if omni_file_dependencydata objects should be resolved to retrieve the dependent object the dependency is pointing to. Dependencies are resolved recursively, i.e., if a dependency points to another dependency then that one will be resolved as well. Default value is False, i.e., the omni_file_dependency object itself is returned.

Returns h5py.Group, h5py.File, or h5py.Dataset corresponding to the given omni_object.

Raises ValueError A ValueError is raised in case that an unsupported omni_object object is given, i.e., the input object is not a omni_file API object nor a h5py Group, File, or Dataset object.

get_managed_group()

Return the h5py object with the analysis data.

The returned object can be used to read data directly from the HDF5 file. Write operations to the analysis group can be performed only if the associated omni_file was opened with write permissions.

Returns h5py object for the analysis group.

classmethod get_num_items(file_group, basename='')

Get the number of object with the given basename at the given path

Parameters

• file_group – The h5py object to be examined
• basename – The name that should be searched for.

Returns Number of objects with the given basename at the given path

classmethod get_omsi_object(h5py_object, resolve_dependencies=False)

This static method is convenience function used to retrieve the corresponding interface class for a given h5py group object.

Parameters

• h5py_object – h5py object for which the corresponding omni_file API object should be generated. This may also be a string describing the requested object based on a combination of the path to the file and a path of the object <filename.h5>:<object_path>
• **resolve_dependencies** – Set to True if omsi_file_dependencydata objects should be resolved to retrieve the dependent object the dependency is pointing to. Dependencies are resolved recursively, i.e., if a dependency points to another dependency then that one will be resolved as well. Default value is False, i.e., the omis_file_dependency object itself is returned.

**Returns**

None in case no corresponding object was found. Otherwise an instance of:

• omsi_file : If the given object is a h5py.File object
• omsi_file_experiment : If the given object is an experiment group
• omsi_file_methods : If the given object is a method group
• omsi_file_instrument : If the given object is an instrument group
• omsi_file_analysis : If the given object is an analysis group
• omsi_file_msidata : If the given object is a MSI data group
• omsi_file_dependencydata : If the given object is a dependency group
• The input h5py_object: If the given object is a h5py.Dataset or h5py.Group
• None: In case that an unknown type is given.

**get_timestamp()**

Get the timestamp when the analysis group was created in the HDF5 file.

**Returns** Python timestamp string generated using time.ctime(). None may be returned in case that the timestamp does not exists or cannot be retrieved from the file for some reason.

**get_version()**

Get the omis version for the representation of this object in the HDF5 file

**classmethod is_managed(in_object)**

Check whether the given object is managed by any omis API class.

**Parameters**

- **in_object** *(Any omis_file API object or h5py.Dataset or h5py.Group or h5py.File object.)* – The object to be checked

**items()**

Get the list of items associdated with the h5py.Group object managed by this object

**static parse_path_string(path)**

Given a string of the form <filename.h5>:<object_path> retrieve the name of the file and the object path.

**Parameters**

- **path** – The string defining the file and object path.

**Returns** Tuple with the filename and the object path. Both may be None depending on whether an object_path is given and whether the path string is valid.

**Raises** ValueError in case that an invalid string is given

**static same_file(filename1, filename2)**

Check whether two files are the same.

This function uses the os.path.samefile(...) method to compare files and falls back to comparing the absolute paths of files if samefile should fail or cannot be imported.

**Parameters**

- **filename1** – The name of the first file
- **filename2** – The name of the second file
Returns

class omsi.dataformat.omsi_file.common.omsi_file_object_manager(*args, **kwargs)
    Bases: object

    Base class used to define manager helper classes used to manage contained managed objects. Managed objects are HDF5.Groups (or Datasets) with a corresponding manager API class and may be nested within other Managed objects.

    What is a manager helper class?

    Manager classes are used in the design of omsi.dataformat.omsi_file to encapsulate functionality needed for management of other manager objects. The expected use of this class, hence, is through multiple inheritance where the main base class is omsi.dataformat.omsi_file.common.omsi_file_common. This is important due to the use of super to accomodate multiple inheritance to allow object to manage an arbitrary number of other object and inherit from other object as well.

    Use of super()

    This class inherits only from object but calls super in the __init__(manager_group) with the manager_group as only input parameter, in the expectation that this class is used using multiple inheritance with omsi_file_common as main base class.

    Multiple inheritance is used in omsi.dataformat.omsi_file module when a class contains other managed objects and uses the manager classes (such as this one) to get all the features needed to manage those objects.

    All child classes of omsi_file_common call super(..).__init__(manager_group) and all manager helper classes (such as this one) use a single input parameter indicating the manager h5py.Group object that contains the given object.

main_file Module

Module for managing OpenMSI HDF5 data files.

class omsi.dataformat.omsi_file.main_file.omsi_file(filename, mode='a', **kwargs)
    Bases: omsi.dataformat.omsi_file.experiment.omsi_experiment_manager,
              omsi.dataformat.omsi_file.common.omsi_file_common

    API for creating and managing a single OpenMSI data file.

    Use of super()

    This class inherits from omsi.dataformat.omsi_file_common.omsi_file_common. Consistent with the design pattern for multiple inheritance of the omsi.dataformat.omsi_file module, the __init__ function calls super(...).__init__(manager_group) with a single parameter indicating the parent group.

Inherited Instance Variables

    Variables

        • managed_group – The group that is managed by this object
        • name – Name of the managed group

Open the given file or create it if does not exit.

The creation of the object may fail if the file does not exist, and the selected mode is ‘r’ or ‘r+’.

Keyword arguments:

    Parameters
• **filename** – string indicating the name+path of the OpenMSI data file. Alternatively this may also be an h5py.File instance (or an h5py.Group, h5py.Dataset instance from which we can get the file)

• **mode** – read/write mode. One of:
  - `r` = readonly, file must exist.
  - `r+` = read/write, file must exist.
  - `w` = Create file, truncate if exists.
  - `w-` = create file, fail if exists.
  - `a` = read/write if exists, create otherwise (default)

• **kargs** – Other keyword arguments to be used for opening the file using h5py. See the h5py.File documentation for details. For example to use parallel HDF5, the following additional parameters can be given `driver='mpio', comm:MPI.COMM_WORLD`.

```python
close_file()
Close the msi data file

flush()
Flush all I/O

get_filename()
Get the name of the omsi file

    Returns String indicating the filename (possibly including the full path, depending on how the object has been initialized)

get_h5py_file()
Get the h5py object for the omsi file

    Returns h5py reference to the HDF5 file

classmethod is_valid_dataset(name)
Perform basic checks for the given filename, whether it is a valid OMSI file.

    Parameters name – Name of the file to be checked.

    Returns Boolean indicating whether the file is valid

write_xdmf_header(xdmf_filename)
Write XDMF header file for the current HDF5 datafile

    Parameters xdmf_filename – The name of the xdmf XML header file to be created for the HDF5 file.
```

**experiment Module**

OMSI file module for management of experiment data.

```python
class omsi.dataformat.omsi_file.experiment.omsi_experiment_manager(experiment_parent)
    Bases: omsi.dataformat.omsi_file.common.omsi_file_object_manager

    Experiment manager helper class used to define common functionality needed for experiment-related data. Usually, a class that defines a format that contains an omsi_file_experiment object will inherit from this class (in addition to omsi_file_common) to acquire the common features.

    For more details see: omsi.dataformat.omsi_file.omsi_common.omsi_file_object_manager
```

10.1. Subpackages
Variables `experiment_parent` – The h5py.Group parent object containing the instrument object to be managed.

`create_experiment(exp_identifier=None, flush_io=True)`

Create a new group in the file for a new experiment and return the `omsi_file_experiment` object for the new experiment.

Parameters

- `exp_identifier (string or None (default))` – The string used to identify the analysis
- `flush_io` – Call flush on the HDF5 file to ensure all HDF5 buffers are flushed so that all data has been written to file.

`get_experiment(exp_index)`

Get the `omsi_format_experiment` object for the experiment with the given index.

Parameters `exp_index (uint)` – The index of the requested experiment

Returns h5py reference to the experiment with the given index. Returns None in case the experiment does not exist.

`get_experiment_by_identifier(exp_identifier_string)`

Get the `omsi_format_experiment` object for the experiment with the given identifier.

Parameters `exp_identifier_string (string)` – The string used to identify the analysis

Returns Returns h5py object of the experiment group or None in case the experiment is not found.

`static get_experiment_path(exp_index=None)`

Based on the index of the experiment return the full path to the hdf5 group containing the data for an experiment.

Parameters `exp_index` – The index of the experiment.

Returns String indicating the path to the experiment.

`get_num_experiments()`

Get the number of experiments in this file.

Returns Integer indicating the number of experiments.

class `omsi.dataformat.omsi_file.experiment.omsi_file_experiment(exp_group)`


Class for managing experiment specific data

Use of `super()`:

This class inherits from `omsi.dataformat.omsi_file.common.omsi_file_common`. Consistent with the design pattern for multiple inheritance of the `omsi.dataformat.omsi_file` module, the `__init__` function calls `super(...).__init__(manager_group)` with a single parameter indicating the parent group.

Inherited instance variable:

Variables

- `managed_group` – The group that is managed by this object
• **methods_parent** – The parent group containing the methods object (same as managed_group)

• **instrument_parent** – The parent group containing the instrument object (same as managed_group)

• **name** – Name of the managed group

Initialize the experiment object given the h5py object of the experiment group

**Parameters**

- **exp_group** – The h5py object with the experiment group of the omsi hdf5 file.

**get_experiment_identifier()**

Get the HDF5 dataset with the identifier description for the experiment.

**Returns**

- h5py object of the experiment identifier or None in case not present

**get_experiment_index()**

Determine the index of the experiment based on the name of the group.

**get_instrument_info**(check_parent=False)

Inherited from omsi_instrument_manager parent class. Overwritten here to change the default parameter setting for check_parent. See omsi.dataformat.omsi_file.instrument.omsi_instrument_manager for details.

**get_method_info**(check_parent=False)

Inherited from omsi_method_manager parent class. Overwritten here to change the default parameter setting for check_parent. See omsi.dataformat.omsi_file.methods.omsi_method_manager for details.

**set_experiment_identifier**(identifier)

Overwrite the current identifier string for the experiment with the given string

**Parameters**

- **identifier** – The new experiment identifier string.

---

**metadata_collection Module**

Module for management of general metadata storage entities. These are often specialized —e.g., omsi_file_instrument, omsi_file_sample—to store specific metadata and add more functionality.

**class** omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection(metadata_group)

**Bases:** omsi.dataformat.omsi_file.common.omsi_file_common

Class for managing method specific data.

**Use of super():**

This class inherits from omsi.dataformat.omsi_file.common.omsi_file_common. Consistent with the design pattern for multiple inheritance of the omsi.dataformat.omsi_file module, the __init__ function calls super(...)__init__(manager_group) with a single parameter indicating the parent group.

**Inherited Instance Variables**

**Variables**

- **managed_group** – The group that is managed by this object

- **name** – Name of the managed group

Initialize the metadata collection object given the h5py object of the metadata collection

**Parameters**

- **metadata_group** – The h5py object with the metadata collection group of the omsi hdf5 file.
add_metadata(metadata)
Add a new metadata entry

Parameters
metadata – Instance of omsi.shared.metadata_data.metadata_value or describing
omsi.shared.metadata_data.metadata_dict with the metadata to be added.

get_metadata(key=None)
Get dict with the full description of the metadata for the given key or all metadata if no key is given.

Returns
omsi.shared.metadata_data.metadata_value object if a key is given or a
omsi.shared.metadata_data.metadata_dict with all metadata if no key is specified.

Raises
KeyError is raised in case that the specified key does not exist

keys()
Get a list of all metadata keys

Returns
List of string with the metadata keys

values()
Convenience function returning a list of all metadata objects. This is equivalent
get_metadata(key=None).values(), however, for consistency with other dict-like interfaces this
function returns a list of omsi.shared.metadata_data.metadata_value objects rather than the
omsi.shared.metadata_data.metadata_dict

Returns
List of omsi.shared.metadata_data.metadata_value with all metadata

class omsi.dataformat.omsi_file.metadata_collection.omsi_metadata_collection_manager
(metadata_parent=None)
Bases: omsi.dataformat.omsi_file.common.omsi_file_object_manager

This is a file format manager helper class used to define common functionality needed for management of
metadata-related data. Usually, a class that defines a format that contains an omsi_file_metadata object will
inherit from this class (in addition to omsi_file_common) to acquire the common features.

For more details see: omsi.dataforamt.omsi_file.omsi_common.omsi_file_object_manager

Variables
metadata_parent – The parent h5py.Group object containing the method object to
be managed

create_metadata_collection(group_name=None, metadata=None, flush_io=True)
Add a new group for managing metadata

Parameters

• group_name (str, None) – Optional name of the new metadata group. If None is given
then the omsi_format_metadata_collection.metadata_collection_groupname_default will
be used

• metadata (None, omsi.shared.metadata_data.metadata_value,
omsi_shared.metadata_data.metadata_dict) – Additional metadata to be added to
the collection after creation

• flush_io – Call flush on the HDF5 file to ensure all HDF5 buffers are flushed so that
all data has been written to file

Returns omsi_file_metadata_collection object

get_default_metadata_collection(omsi_object=None)
Get the default metadata collection object if it exists

Parameters
omsi_object – The omsi file API object or h5py.Group object that we should
check. If set to None (default) then the self.metadata_parent will be used

Returns
None, omsi_file_metadata_collection
get_metadata_collections (omsi_object=None, name=None)
Get all metadata_collections defined for given OpenMSI file API object or h5py.Group.

Parameters

• omsi_object – The omsi file API object or h5py.Group object that we should check. If set to None (default) then the self.metadata_parent will be used

• name – If name is specified, then only retrieve collections with the given name

Returns List of omsi_file_metadata_collection objects for the requested group. The function returns None in case that the h5py.Group for the omsi_object could not be determined.

has_default_metadata_collection (omsi_object)
Check whether the omsi API object (or h5py.Group) contains any a metadata collection with the default name.

Returns bool

has_metadata_collections (omsi_object=None)
Check whether the given omsi API object (or h5py.Group) contains any metadata collections

Parameters omsi_object – The omsi file API object or h5py.Group object that we should check. If set to None (default) then the self.metadata_parent will be used

Returns Boolean indicating whether metadata collections were found

instrument Module
Module for managing instrument related data in OMSI files.

class omsi.dataformat.omsi_file.instrument.omsi_file_instrument (instrument_group)
Bases: omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection

Class for managing instrument specific data

Use of super():
This class inherits from omsi.dataformat.omsi_file.common.omsi_file_common. Consistent with the design pattern for multiple inheritance of the omsi.dataformat.omsi_file module, the __init__ function calls super(...).__init__(manager_group) with a single parameter indicating the parent group.

Inherited Instance Variables

Variables

• managed_group – The group that is managed by this object

• name – Name of the managed group

Initialize the instrument object given the h5py object of the instrument group

Parameters instrument_group – The h5py object with the instrument group of the omsi hdf5 file.

get_instrument_mz ()
Get the HDF5 dataset with the mz data for the instrument.

To get the numpy array of the full mz data use: get_instrument_mz()[:]

Returns Returns the h5py object with the instrument mz data. Returns None in case no mz data was found for the instrument.

10.1. Subpackages
get_instrument_name()
Get the HDF5 dataset with the name of the instrument.

To get the string of the instrument name use: get_instrument_name()[...]

Returns h5py object to the dataset with the instrument name. Returns None in case no method
name is found.

has_instrument_name()
Check whether a name has been saved for the instrument

Returns bool

set_instrument_name(name)
Overwrite the current identifier string for the experiment with the given string.

Parameters name (string.) – The new instrument name.

class omsi.dataformat.omsi_file.instrument.omsi_instrument_manager (instrument_parent)
Bases: omsi.dataformat.omsi_file.metadata_collection.omsi_metadata_collection_manager

Instrument manager helper class used to define common functionality needed for instrument-related data. Usu-
ally, a class that defines a format that contains an omsi_file_methods object will inherit from this class (in
addition to omsi_file_common) to acquire the common features.

For more details see: omsi.dataforamt.omsi_file.omsi_common.omsi_file_object_manager

Variables instrument_parent – The h5py.Group parent object containing the instrument ob-
ject to be managed.

create_instrument_info (instrument_name=None, mzdata=None, flush_io=True)
Add information about the instrument used for creating the images for this experiment.

Parameters
• instrument_name (string, None) – The name of the instrument
• mzdata (numpy array or None) – Numpy array of the mz data values of the instrument
• flush_io – Call flush on the HDF5 file to ensure all HDF5 bufferes are flushed so that
  all data has been written to file

Returns The function returns the h5py HDF5 handler to the instrument info group created for
the experiment.

get_instrument_info (check_parent=True)
Get the HDF5 group object with the instrument information.

Parameters check_parent – If no method group is available for this dataset should we check
whether the parent object (i.e., the experiment group containing the dataset) has information
about the method. (default=True)

Returns omsi_file_instrument object for the requested instrument info. The function returns
None in case no instrument information was found for the experiment

has_instrument_info (check_parent=False)
Check whether custom instrument information is available for this dataset.

Parameters check_parent – If no instrument group is available for this dataset should we
check whether the parent object (i.e., the experiment group containing the dataset) has inform-
ation about the instrument. (default=False)

Returns Boolean indicating whether instrument info is available.
methods Module

Module for management of method specific data in OMSI data files

class omsi.dataformat.omsi_file.methods.omsi_file_methods(method_group)
    Bases: omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection

Class for managing method specific data.

Use of super():

This class inherits from omsi.dataformat.omsi_file.common.omsi_file_common. Consistent with the design pattern for multiple inheritance of the omsi.dataformat.omsi_file module, the __init__ function calls super(...).__init__(manager_group) with a single parameter indicating the parent group.

Inherited Instance Variables

Variables

    • managed_group – The group that is managed by this object
    • name – Name of the managed group

Initialize the method object given the h5py object of the method group

Parameters method_group – The h5py object with the method group of the omsi hdf5 file.

get_method_name()  
Get the HDF5 dataset with the name of the method.

To retrieve the name string use get_method_name()[...]

    Returns h5py object where the method name is stored. Returns None in case no method name is found.

has_method_name()  
Check whether an object has a method name

    Returns bool

set_method_name(name_string)  
Overwrite the name string for the method with the given name string

    Parameters name_string (string) – The new method name.

class omsi.dataformat.omsi_file.methods.omsi_methods_manager(methods_parent=None)
    Bases: omsi.dataformat.omsi_file.metadata_collection.omsi_metadata_collection_manager

This is a file format manager helper class used to define common functionality needed for methods-related data. Usually, a class that defines a format that contains an omsi_file_methods object will inherit from this class (in addition to omsi_file_common) to acquire the common features.

For more details see: omsi.dataforamt.omsi_file.omsi_common.omsi_file_object_manager

    Variables method_parent – The parent h5py.Group object containing the method object to be managed

create_method_info(method_name=None, metadata=None, flush_io=True)
Add information about the method imaged to the experiment. Note, if a methods group already exists, then that group will be used. If method_name is not None, then the existing name will be overwritten by the new value.

    Parameters

        • method_name (str, None) – Optional name of the method
• **metadata** (*metadata_value, metadata_dict*) – Additional metadata to be stored with the methods

• **flush_io** – Call flush on the HDF5 file to ensure all HDF5 buffers are flushed so that all data has been written to file

**Returns** h5py object of the newly created method group.

**get_method_info** (*check_parent=True*)

Get the `omsi_file_methods` object with the method information.

**Parameters**

- **check_parent** – If no method group is available for this dataset should we check whether the parent object (i.e., the experiment group containing the dataset) has information about the method. (default=True)

**Returns** `omsi_file_methods` object for the requested method info. The function returns None in case no method information was found for the experiment

**has_method_info** (*check_parent=False*)

Check whether custom method information is available for this dataset.

**Parameters**

- **check_parent** – If no method group is available for this dataset should we check whether the parent object (i.e., the experiment group containing the dataset) has information about the method. (default=False)

**Returns** Boolean indicating whether method info is available.

**msidata Module**

Module for managing MSI data in OMSI data files

```python
class omsi.dataformat.omsi_file.msidata.omsi_file_msidata(data_group, fill_space=True, fill_spectra=True, preload_mz=False, preload_xy_index=False)

```

Interface for interacting with mass spectrometry imaging datasets stored in omis HDF5 files. The interface allows users to interact with the data as if it were a 3D cube even if data is missing. Full spectra may be missing in cases where only a region of interest in space has been imaged. Spectra may further be pre-processed so that each spectrum has only information about its peaks so that each spectrum has its own mz-axis.

To load data use standard array syntax, e.g., `[1,1,:)` can be used to retrieve the spectrums at location (1,1).

**Use of super():**

This class inherits from `omsi.dataformat.omsi_file.common.omsi_file_common`. Consistent with the design pattern for multiple inheritance of the `omsi.dataformat.omsi_file` module, the `__init__` function calls `super(...).__init__(manager_group)` with a single parameter indicating the parent group.

**Current limitations:**

- The estimates in `def __best_dataset__(self,keys)` are fairly crude at this point
- The `__getitem__` function for the partial_spectra case is not implemented yet.
• The \_\_setitem\_ function for the partial spectra case is not implemented yet (Note, it should also support dynamic expansion of the cube by adding previously missing spectra).

• For the partial cube case, assignment using \_\_setitem\_ function is only supported to valid spectra, i.e., spectra that were specified as occupied during the initial creation process.

Public object variables:

Variables

• shape – Define the full 3D shape of the dataset (i.e., even if the data is stored in sparse manner)

• dtype – The numpy datatype of the main MSI data. This is the same as dataset.dtype

• name – The name of the corresponding group in the HDF5 file. Used to generate hard-links to the group.

• format_type – Define according to which standard the data is stored in the file

• datasets – List of h5py objects containing possibly multiple different version of the same MSI data (spectra). There may be multiple versions stored with different layouts in order to optimize the selection process.

• mz – dataset with the global mz axis information. If prelaod_mz is set in the constructor, then this is a numpy dataset with the preloaded data. Otherwise, this is the h5py dataset pointing to the data on disk.

• xy\_index – None if format_type is ‘full\_cube’. Otherwise, this is the 2D array indicating for each x/y location the index of the spectrum in dataset. If prelaod_xy\_index is set in the constructor, then this is a numpy dataset with the preloaded data. Otherwise, this is the h5py dataset pointing to the data on disk. Negative (-1) entries indicate that no spectrum has been recorded for the given pixel.

• inv_xy\_index – 2D dataset with n rows and 2 columns indicating for each spectrum i the (x,y) pixel index the spectrum belongs to. This index is stored for convenience purposes but is not actually needed for data access.

• mz\_index – None if format_type is not ‘partial\_spectra’. Otherwise this is a dataset of the same size as the spectra data stored in dataset. Each entry indicates an index into the mz dataset to determine the mz\_data value for a spectrum. This means mz[ mx\_index ] gives the true mz value.

• xy\_index\_end – None if format_type is not ‘partial\_spectra’. Otherwise this is a 2D array indicating for each x/y location the index where the given spectrum ends in the dataset. If prelaod_xy\_index is set in the constructor, then this is a numpy dataset with the preloaded data. Otherwise, this is the h5py dataset pointing to the data on disk. Negative (-1) entries indicate that no spectrum has been recorded for the given pixel.

Private object variables:

Variables

• \_data\_group – Store the pointer to the HDF5 group with all the data

• \_fill\_xy – Define whether the data should be reconstructed as a full image cube Set using the set\_fill\_space function(…)

• \_fill\_mz – Define whether spectra should be remapped onto a global m/z axis. Set using the set\_fill\_spectra function(…)

Initialize the omsi\_msidata object.
The fill options are provided to enable a more convenient access to the data independent of how the data is stored in the file. If the fill options are enabled, then the user can interact with the data as if it were a 3D cube while missing is data is filled in by the given fill value.

The preload options provided here refer to generally smaller parts of the data for which it may be more efficient to load the data and keep it around rather than doing repeated reads. If the object is used only for a single read and destroyed afterwards, then disabling the preload options may give a slight advantage but in most cases enabling the preload should be Ok (default).

**Parameters**

- **data_group** – The h5py object for the group with the omsi_msidata.
- **fill_space** – Define whether the data should be padded in space (filled with 0’s) when accessing the data using [..] operator so that the data behaves like a 3D cube.
- **fill_spectra** – Define whether the spectra should completed by adding 0’s so that all spectra retrieved via the [..] operator so that always spectra of the full length are returned. This option is provided to ease extension of the class to cases where only partial spectra are stored in the file but is not used at this point.
- **preload_mz** – Should the data for the mz axis be kept in memory or loaded on the fly when needed.
- **preload_xy_index** – Should the xy index (if available) be preloaded into memory or should the required data be loaded on the fly when needed.

**copy_dataset** *(source, destination, print_status=False)*

Helper function used to copy a source msi dataset one chunk at a time to the destination dataset. The data copy is done one destination chunk at a time to achieve chunk-aligned write.

**Parameters**

- **source** – The source h5py dataset
- **destination** – The h5py destination h5py dataset.
- **print_status** – Should the function print the status of the conversion process to the command line?

**create_optimized_chunking** *(chunks=None, compression=None, compression_opts=None, copy_data=True, print_status=False, flush_io=True)*

Helper function to allow one to create optimized copies of the dataset with different internal data layouts to speed up selections. The function expects that the original data has already been written to the data group.

The function takes

**Parameters**

- **chunks** – Specify whether chunking should be used (True,False), or specify the chunk sizes to be used explicitly.
- **compression** – h5py compression option. Compression strategy. Legal values are ‘gzip’, ‘szip’, ‘lz4’. Can also use an integer in range(10) indicating gzip.
- **compression_opts** – h5py compression settings. This is an integer for gzip, 2-tuple for szip, etc.. For gzip (H5 deflate filter) this is the aggression parameter. The aggression parameter is a number between zero and nine (inclusive) to indicate the tradeoff between speed and compression ratio (zero is fastest, nine is best ratio).
- **copy_data** – Should the MSI data be copied by this function to the new dataset or not. If False, then it is up to the user of the function to copy the appropriate data into the returned h5py dataset (not recommended but may be useful for performance optimization).
• **print_status** – Should the function print the status of the conversion process to the command line?

• **flush_io** – Call flush on the HDF5 file to ensure all HDF5 buffers are flushed so that all data has been written to file

**Returns** h5py dataset with the new copy of the data

### get_h5py_datasets(index=0)
Get the h5py dataset object for the given dataset.

**Parameters**
- **index** – The index of the dataset.

**Returns** h5py object for the requested dataset.

**Raises** and Index error is generated in case an invalid index is given.

### get_h5py_mzdata()
Get the h5py object for the mz datasets.

**Returns** h5py object of the requested mz dataset.

### set_fill_space(fill_space)
Define whether spatial selection should be filled with 0’s to retrieve full image slices

**Parameters**
- **fill_space** – Boolean indicating whether images should be filled with 0’s

### set_fill_spectra(fill_spectra)
Define whether spectra should be filled with 0’s to map them to the global mz axis when retrieved.

**Parameters**
- **fill_spectra** – Define whether m/z values should be filled with 0’s.

### class omsi.dataformat.omsi_file.msidata.omsi_msidata_manager(msidata_parent)
Bases: omsi.dataformat.omsi_file.common.omsi_file_object_manager

MSI-data manager helper class used to define common functionality needed for msidata-related data. Usually, a class that defines a format that contains an omsi_file_msidata object will inherit from this class (in addition to omsi_file_common) to acquire the common features.

For more details see: omsi.dataforamt.omsi_file.omsi_common.omsi_file_object_manager

**Variables**
- **msidata_parent** – The h5py.Group parent object containing the instrument object to be managed.

Initialize the manager object.

**Parameters**
- **msidata_parent** – The h5py.Group parent object for the msi data.

### create_msidata_full_cube(data_shape, data_type='f', mzdata_type='f', chunks=None, compression=None, compression_opts=None, flush_io=True)
Create a new mass spectrometry imaging dataset for the given experiment written as a full 3D cube.

**Parameters**
- **data_shape** – Shape of the dataset. Eg. shape=(10,10,10) creates a 3D dataset with 10 entries per dimension
- **data_type** – numpy style datatype to be used for the dataset.
- **mzdata_type** – numpy style datatype to be used for the mz data array.
- **chunks** – Specify whether chunking should be used (True,False), or specify the chunk sizes to be used in x,y, and m/z explicitly.
- **compression** – h5py compression option. Compression strategy. Legal values are ‘gzip’, ‘szip’, ‘lzf’. Can also use an integer in range(10) indicating gzip.
create_msidata_partial_cube(
    data_shape, mask, data_type='f', mzdata_type='f',
    chunks=None, compression=None, compression_opts=None,
    flush_io=True)

Create a new mass spectrometry imaging dataset for the given experiment written as a partial 3D cube of complete spectra.

Parameters

• **data_shape** – Shape of the dataset. Eg. shape=(10,10,10) creates a 3D dataset with 10 entries per dimension
• **mask** – 2D boolean NumPy array used as mask to indicate which (x,y) locations have spectra associated with them.
• **data_type** – numpy style datatype to be used for the dataset.
• **mzdata_type** – numpy style datatype to be used for the mz data array.
• **chunks** – Specify whether chunking should be used (True, False), or specify the chunk sizes to be used in x,y, and m/z explicitly.
• **compression** – h5py compression option. Compression strategy. Legal values are ‘gzip’, ‘szip’, ‘lzf’. Can also use an integer in range(10) indicating gzip.
• **compression_opts** – h5py compression settings. This is an integer for gzip, 2-tuple for szip, etc.. For gzip (H5 deflate filter) this is the aggression parameter. The aggression parameter is a number between zero and nine (inclusive) to indicate the tradeoff between speed and compression ratio (zero is fastest, nine is best ratio).
• **flush_io** – Call flush on the HDF5 file to ensure all HDF5 bufferes are flushed so that all data has been written to file

Returns

The following two empty (but appropriately sized) h5py datasets are returned in order to be filled with data:

• **data_dataset** : Primary h5py dataset for the MSI data with shape data_shape and dtype data_type.
• **mz_dataset** : h5py dataset for the mz axis data with shape [data_shape[2]] and dtype mzdata_type.

Returns **data_group**: The h5py object with the group in the HDF5 file where the data should be stored.
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

• mz_dataset: h5py dataset for the mz axis data with shape [data_shape[2]] and dtype mzdata_type.

Returns
The following already complete dataset
• xy_index_dataset: This dataset indicates for each xy location to which index in data_dataset the location corresponds to. This dataset is needed to identify where spectra need to be written to.

Returns data_group: The h5py object with the group in the HDF5 file where the data should be stored.

create_msidata_partial_spectra(spectra_length, len_global_mz, data_type='f', mzdata_type='f', chunks=None, compression=None, compression_opts=None, flush_io=True)
Create a new mass spectrometry imaging dataset for the given experiment written as a partial 3D cube of partial spectra.

Parameters
• spectra_length – 2D boolean NumPy array used indicating for each (x,y) locations the length of the corresponding partial spectrum.
• len_global_mz – The total number of m/z values in the global m/z axis for the full 3D cube
• data_type – The dtype for the MSI dataset
• mzdata_type – The dtype for the mz dataset
• mzdata_type – numpy style datatype to be used for the mz data array.
• chunks – Specify whether chunking should be used (True, False), or specify the chunk sizes to be used in x,y, and m/z explicitly.
• compression – h5py compression option. Compression strategy. Legal values are ‘gzip’, ‘szip’, ‘lz4’. Can also use an integer in range(10) indicating gzip.
• compression_opts – h5py compression settings. This is an integer for gzip, 2-tuple for szip, etc.. For gzip (H5 deflate filter) this is the aggression parameter. The aggression parameter is a number between zero and nine (inclusive) to indicate the tradeoff between speed and compression ratio (zero is fastest, nine is best ratio).
• flush_io – Call flush on the HDF5 file to ensure all HDF5 bufferes are flushed so that all data has been written to file

Returns
The following two empty (but approbriatelus sized) h5py datasets are returned in order to be filled with data:
• data_dataset: The primary h5py dataset for the MSI data with shape data_shape and dtype data_type.
• mz_index_dataset: h5py dataset with the mz_index values
• mz_dataset: h5py dataset for the mz axis data with shape [data_shape[2]] and dtype mzdata_type.

Returns
The following already complete dataset

10.1. Subpackages
• **xy_index_dataset** [This dataset indicates for each xy location at which index in data_dataset the corresponding spectrum starts. This dataset is needed to identify where spectra need to be written to.]

• **xy_index_end_dataset** [This dataset indicates for each xy location at which index in data_dataset the corresponding spectrum ends (excluding the given value). This dataset is needed to identify where spectra need to be written to.]

**Returns** data_group: The h5py object with the group in the HDF5 file where the data should be stored.

**get_msidata** *(data_index, fill_space=True, fill_spectra=True, preload_mz=True, preload_xy_index=True)*

Get the dataset with the given index for the given experiment.

For more detailed information about the use of the fill_space and fill_spectra and preload_mz and preload_xy_index options, see the init function of omsi.dataformat.omsi_file_msidata.

**Parameters**

• **data_index** *(unsigned int)* – Index of the dataset.

• **fill_space** – Define whether the data should be padded in space (filled with 0’s) when accessing the data using [..] operator so that the data behaves like a 3D cube.

• **fill_spectra** – Define whether the spectra should completed by adding 0’s so that all spectra retrieved via the [..] operator so that always spectra of the full length are returned.

• **preload_mz** – Should the data for the mz axis be kept in memory or loaded on the fly when needed.

• **preload_xy_index** – Should the xy index (if available) be preloaded into memory or should the required data be loaded on the fly when needed.

**Returns** omsi_file_msidata object for the given data_index or None in case the data with given index does not exist or the access failed for any other reason.

**get_msidata_by_name** *(data_name)*

Get the h5py data object for the the msidata with the given name.

**Parameters** data_name *(string)* – The name of the dataset

**Returns** h5py object of the dataset or None in case the dataset is not found.

**get_num_msidata** *

Get the number of raw mass spectrometry images stored for a given experiment

**Returns** Integer indicating the number of msi datasets available for the experiment.

**analysis Module**

Module for managing custom analysis data in OMSI HDF5 files.

**class** omsi.dataformat.omsi_file.analysis.omsi_analysis_manager *(analysis_parent)*

**Bases:** omsi.dataformat.omsi_file.common.omsi_file_object_manager

Analysis manager helper class used to define common functionality needed for analysis-related data. Usually, a class that defines a format that contains an omsi_file_analysis object will inherit from this class (in addition to omsi_file_common) to acquire the common features.

For more details see: omsi.dataformat.omsi_file.omsi_common.omsi_file_object_manager
Variables `analysis_parent` – The h5py.Group parent object containing the instrument object to be managed.

`create_analysis` (analysis, flush_io=True, force_save=False, save_unsaved_dependencies=True, mpi_root=0, mpi_comm=None)

Add a new group for storing derived analysis results for the current experiment

Create the analysis group using `omsi_file_analysis.__create__` which in turn uses `omsi_file_analysis.__populate_analysis__(...)` to populate the group with the appropriate data.

NOTE: Dependencies are generally resolved to point to file objects. However, if `save_unsaved_dependencies` is set to False and a given in-memory dependency has not been saved yet, then the value associated with that dependency will be saved instead as part of the parameters and, hence, only the value of the dependency is persevered in that case and not the full dependency chain.

NOTE: Dependencies if they only exists in memory are typically saved recursively unless `save_unsaved_dependencies` is set to False. I.e, calling `create_analysis` may result in the creating of multiple other dependent analyses if they have not been saved before.

Parameters

- `analysis` (omsi.analysis.analysis_base) – Instance of omsi.analysis.analysis_base defining the analysis
- `flush_io` (bool) – Call flush on the HDF5 file to ensure all HDF5 bufferes are flushed so that all data has been written to file
- `force_save` (bool) – Should we save the analysis even if it has been saved in the same location before? If `force_save` is False (default) and the self.omsi_analysis_storage parameter of the analysis object contains a matching storage location—i.e., same file and experiment—, then the analysis will not be saved again, but the object will only be retrieved from file. If `force_save` is True, then the analysis will be saved either way and the self.omsi_analysis_storage parameter will be extended.
- `save_unsaved_dependencies` (bool) – If there are unsaved (in-memory) dependencies, then should those be saved to file as well? Default value is True, i.e, by default all in-memory dependencies that have not been saved yet, i.e, for which the self.omsi_analysis_storage of the corresponding `omsi_analysis__base` object is empty, are saved as well. If in-memory dependencies have been saved before, then a link to those dependencies will be established, rather than re-saving the dependency.
- `mpi_root` – The root MPI process that should perform the writing. This is to allow all analyses to call the function and have communication in the analysis.write_analysis_data function be handled.
- `mpi_comm` – The MPI communicator to be used. None if default should be used (i.e., `MPI.COMM_WORLD`)

Returns The `omsi_file_analysis` object for the newly created analysis group and the integer index of the analysis. NOTE: if `force_save` is False (default), then the group returned may not be new but may be simply the first entry in the list of existing storage locations for the given analysis. NOTE: If we are in MPI parallel and we are on a core that does not write any data, then None is returned instead.

`static create_analysis_static` (analysis_parent, analysis, flush_io=True, force_save=False, save_unsaved_dependencies=True, mpi_root=0, mpi_comm=None)

Same as `create_analysis(...)` but instead of relying on object-level, this function allows additional parameters (specifically the `analysis_parent`) to be provided as input, rather than being determined based on self
Parameters

- **analysis_parent** – The h5py.Group object or omsi.dataformat.omsi_file.common.omsi_file_common object where the analysis should be created.
- **kwargs** – Additional keyword arguments for create_analysis(...). See create_analysis(...) for details.

**Returns** The output of create_analysis

```python
get_analysis(analysis_index)
```
Get the omsi_format_analysis analysis object for the experiment with the given index.

**Parameters**

- **analysis_index** *(Unsigned integer)* – The index of the analysis

**Returns** omsi_file_analysis object for the requested analysis. The function returns None in case the analysis object was not found.

```python
get_analysis_by_identifier(analysis_identifier_string)
```
Get the omsi_format_analysis analysis object for the analysis with the given identifier.

**Parameters**

- **analysis_identifier_string** *(string)* – The string used as identifier for the analysis.

**Returns** h5py object of the analysis or None in case the analysis is not found.

```python
get_analysis_identifiers()
```
Get a list of all identifiers for all analysis stored for the experiment

**Returns** List of strings of analysis identifiers.

```python
get_num_analysis()
```
Get the number of raw mass spectrometry images stored for a given experiment

**Returns** Integer indicating the number of analyses available for the experiment.

```python
class omsi.dataformat.omsi_file.analysis.omsi_file_analysis(analysis_group)
```
Class for managing analysis specific data in omsi hdf5 files

Initialize the analysis object given the h5py object of the analysis group.

**Parameters**

- **analysis_group** – The h5py object with the analysis group of the omsi hdf5 file.

```python
get_all_analysis_data(load_data=False)
```
Get all analysis data associated with the analysis.

**Parameters**

- **load_data** *(load_data=False)* – load_data: Should the data be loaded or just the h5py objects be stored in the dictionary.

**Returns** List of analysis_data objects with the names and h5py or numpy objects. Access using [index]['name'] and [index]['data'].

```python
get_all_parameter_data(load_data=False, exclude_dependencies=False)
```
Get all parameter data associated with the analysis.

**Parameters**

- **load_data** *(load_data=False)* – Should the data be loaded or just the h5py objects be stored in the dictionary.

**Returns** List of parameter_data objects with names and h5py or numpy object. Access using [index]['name'] and [index]['data'].
get_all_runinfo_data (load_data=False)

Get a dict of all runtime information stored in the file

Returns omsi.shared.run_info_data.run_info_dict type python dict with the runtime information restored.

get_analysis_data_names ()

This function returns all dataset names (and groups) that are custom to the analysis, i.e., that are not part of the omsi file standard.

Returns List of analysis-specific dataset names.

get_analysis_data_shapes_and_types ()

This function returns two dictionaries with all dataset names (and groups) that are custom to the analysis, i.e., that are not part of the omsi file standard, and identifies the shape of the analysis data objects.

Returns Dictionary indicating for each analysis-specific dataset its name (key) and shape (value). And a second dictionary indicating the name (key) and dtype of the dataset.

get_analysis_identifier ()

Get the identifier name of the analysis.

Use get_analysis_identifier()[:] to retrieve the identifier string.

Returns h5py object for the dataset with the identifier string. Returns None, in case no identifier exists. This should not be the case for a valid OpenMSI file.

get_analysis_index ()

Based on the name of the group, get the index of the analysis.

Returns Integer index of the analysis in the file.

get_analysis_type ()

Get the type for the analysis.

Use get_analysis_type()[:] to retrieve the type string.

Returns h5py object with the dataset of the analysis string. Returns, None in case no analysis type exists. This should not be the case in a valid omsi file.

recreate_analysis (**kwargs)

Load an analysis from file and re-execute it. This is equivalent to omsi_analysis.base.restore_analysis().execute()

Parameters **kwargs – Additional keyword arguments to be passed to the execute function of the analysis

Returns Instance of the specific analysis object (e.g., omsi_nmf) that inherits from omsi.analysis.analysis_base with the input parameters and dependencies restored from file. The output, however, is the result from re-executing the analysis. None is returned in case the analysis object cannot be created.

restore_analysis (load_data=True, load_parameters=True, load_runtime_data=True, dependencies_omsi_format=True)

Load an analysis from file and create an instance of the appropriate analysis object defined by the analysis type (i.e., a derived class of omsi.analysis.analysis_base)

Parameters

• load_data – Should the analysis data be loaded from file (default) or just stored as h5py data objects

• load_parameters – Should parameters be loaded from file (default) or just stored as h5py data objects.
• **load_runtime_data** – Should runtime data be loaded from file (default) or just stored as h5py data objects.

• **dependencies_omsi_format** – Should dependencies be loaded as omsi_file API objects (default) or just as h5py objects.

**Returns** Instance of the specific analysis object (e.g., omsi_nmf) that inherits from omsi.analysis.analysis_base with the input parameters, output result, and dependencies restored. We can call execute(…) on the returned object to rerun an analysis. May return analysis_generic in case that the specific analysis is not known.

**dependencies Module**

Base module for managing of dependencies between data in OpenMSI HDF5 files

```python
class omsi.dataformat.omsi_file.dependencies.omsi_dependencies_manager(dependencies_parent):
    Bases: omsi.dataformat.omsi_file.common.omsi_file_object_manager

    Dependencies manager helper class used to define common functionality needed for managing dependencies. Usually, a class that defines a format that contains an omsi_file_dependencies object will inherit from this class (in addition to omsi_file_common) to acquire the common features.

    For more details see: omsi.dataformat.omsi_file.omsi_common.omsi_file_object_manager
```

**Variables**

- **dependencies_parent** – h5py.Group object containing the dependencies object(s) to be managed
- **dependencies** – omsi_file_dependencies object managed by this object or None

**Parameters**

- **dependencies_parent** – Parent group containing the dependencies object to be managed

**add_dependency** (dependency, flush_io=True)

Create a new dependency for this dataset

**Parameters**

- **dependency** – omsi.shared.dependency_dict object describing the data dependency
- **flush_io** – Call flush on the HDF5 file to ensure all HDF5 buffers are flushed so that all data has been written to file

**Returns** omsi_file_dependencydata object with the dependency data or None in case that an error occurred and the dependency has not been generated.

```python
def create_dependencies(dependencies_data_list=None):
    Create a managed group for storing data dependencies if none exists and store the given set of dependencies in it. If a self.dependencies object already exists, then the given dependencies will be added.
    This is effectively a shortcut to omsi_file_dependencies.__create___(...) with specific settings for the current dependencies object managed by self.

    **Parameters**
    - **dependencies_data_list** – List of dependency_dict objects to be stored as dependencies. Default is None which is mapped to an empty list []

    **Returns** omsi_file_dependencies object created by the function.
```

**get_all_dependency_data** (omsi_dependency_format=True)

Get all direct dependencies associated with the data object.
This is convenience function providing access to self.dependencies.get_all_dependency_data(...) which is a function of omsi_file_dependencies class.

**Parameters**

- **omsi_dependency_format** – Should the dependencies be retrieved as omsi_analysis_dependency object (True) or as an omsi_file_dependencydata object (False).

**Returns** List dependency_dict objects containing either omsi file API objects or h5py objects for the dependencies. Access using [index][`name`] and [index][`data`].

**get_all_dependency_data_graph**

```python
include_omsi_dependency=False,
include_omsi_file_dependencydata=False,
recursive=True,
level=0,
name_key='name',
prev_nodes=None,
prev_links=None,
parent_index=None,
metadata_generator=None,
metadata_generator_kwargs=None)
```

Get all direct and indirect dependencies associated with the analysis in form of a graph describing all nodes and links in the provenance hierarchy.

This is convenience function providing access to self.dependencies.get_all_dependency_data_graph(...) which is a function of omsi_file_dependencies class.

**Parameters**

- **include_omsi_dependency** – Should the dependency_dict object be included in the entries in the nodes dict?

- **include_omsi_file_dependencydata** – Should the omsi_file_dependencydata object be included in the entries in the nodes dict?

- **recursive** – Should we trace dependencies recursively to construct the full graph, or only the direct dependencies. Default true (ie., trace recursively)

- **name_key** – Which key should be used in the dicts to indicate the name of the object? Default value is `name`

- **level** – Integer used to indicated the recursion level. Default value is 0.

- **prev_nodes** – List of nodes that have been previously generated. Note, this list will be modified by the call. Note, each node is represented by a dict which is expected to have at least the following keys defined, path, name_key, level (name_key refers to the key defined by the input parameter name_key).

- **prev_links** – Previouly established links in the list of nodes. Note, this list will be modified by the call.

- **parent_index** – Index of the parent node in the list of prev_nodes for this call.

- **metadata_generator** – Optional parameter. Pass in a function that generates additional metadata about a given omsi API object. Note, the key’s level and path and name (i.e., name_key) are already set by this function. The metadata_generator may overwrite these key’s, however, the path has to be unique as it is used to identify duplicate nodes. Overwriting the path with a non-unique value, hence, will lead to errors (missing entries) when generating the graph. Note, the metadata_generator function needs to support the following keyword arguments:
  - `inDict` : The dictionary to which the metadata should be added to.
  - `obj` : The omsi file API object for which metadata should be generated
  - `name` : A qualifying name for the object
  - `name_key` : The key to be used for storing the name
• **metadata_generator_kwargs** – Dictionary of additional keyword arguments that should be passed to the metadata_generator function.

**Returns**

Dictionary containing (two lists. 1) nodes [List of dictionaries, describing the elements] in the dependency graph. 2) links : List of tuples with the links in the graph. Each tuple consists of two integer indices for the nodes list. For each node the following entries are given:

• **dependency_dict**: Optional key used to store the corresponding dependency_dict object. Used only if include_omsi_dependency is True.

• **omsi_file_dependencydata**: Optional key used to store the corresponding omsi_file_dependencydata object. Used only if include_omsi_file_dependencydata is True.

• **name**: Name of the dependency. The actual key is specified by name_key

• **level**: The recursion level at which the object occurs.

• **...**: Any other key/value pairs from the dependency_dict dict.

get_all_dependency_data_recursive (**omsi_dependency_format=True,**
                                     **omsi_main_parent=None,**
                                     **dependency_list=None**)

Get all direct and indirect dependencies associated with the data object.

This is a convenience function providing access to self.dependencies.get_all_dependency_data_recursive(...) which is a function of omsi_file_dependencies class.

**NOTE**: omsi_main_parent and omsi_main_parent are used primarily to ensure that the case of circular dependencies are supported properly. Circular dependencies may occur in the case of semantic dependencies (rather than pure use dependencies), e.g., two datasets that are related modalities may reference each other, e.g., MS1 pointing to related MS2 data and the MS2 datasets referencing the corresponding MS1 datasets.

**Parameters**

• **omsi_dependency_format** – Should the dependencies be retrieved as dependency_dict object (True) or as an omsi_file_dependencydata object (False)

• **omsi_main_parent** – The main parent for which the dependencies are calculated. This is needed to avoid recursion back into the main parent for which we are computing dependencies and avoiding that it is added itself as a dependency for itself. If set to None, then we will use our own self.dependencies_parent object

• **dependency_list** – List of previously visited/created dependencies. This is needed only to avoid deep recursion and duplication due to circular dependencies

**Returns** List analysis_data objects containing either omsi file API interface objects or h5py objects for the dependencies. Access using [index][‘name’] and [index][‘data’].

has_dependencies()

Check whether any dependencies exists for this datasets.

class omsi.dataformat.omsi_file.dependencies.omsi_file_dependencies(**dependencies_group**)

**Bases**: omsi.dataformat.omsi_file.common.omsi_file_common

Class for managing collections of dependencies.

** Use of super()**
This class inherits from `omsi.dataformat.omsi_file.common.omsi_file_common`. Consistent with the design pattern for multiple inheritance of the `omsi.dataformat.omsi_file` module, the `__init__` function calls `super(...).__init__(manager_group)` with a single parameter indicating the parent group.

```python
static_omsi_file_dependencies__create_dependency_graph_node
```

Internal helper function used to create a new node in the graph

**Parameters**

- **level** – The recursion level at which the node exists
- **name** – The name of the node
- **path** – The path of the node
- **dependency_object** – The `omsi_file_dependencydata` object. May be None in case a node to a specific object is set
- **omsi_object** – The OpenMSI file API object. This is required and may NOT be None.
- **include_omsi_dependency** – Should the dependency_dict object be included in the entries in the nodes dict?
- **include_omsi_file_dependencydata** – Should the `omsi_file_dependencydata` object be included in the entries in the nodes dict?
- **name_key** – Which key should be used in the dicts to indicate the name of the object? Default value is 'name'
- **metadata_generator** – Optional parameter. Pass in a function that generates additional metadata about a given omsi API object. Note, the key's level and path and name (i.e., name_key) are already set by this function. The metadata_generator may overwrite these key's, however, the path has to be unique as it is used to identify duplicate nodes. Overwriting the path with a non-unique value, hence, will lead to errors (missing entries) when generating the graph. Note, the metadata_generator function needs to support the following keyword arguments:
  - `in_dict`: The dictionary to which the metadata should be added to.
  - `obj`: The omsi file API object for which metadata should be generated
  - `name`: A qualifying name for the object
  - `name_key`: The key to be used for storing the name
- **metadata_generator_kwargs** – Dictionary of additional keyword arguments that should be passed to the metadata_generator function.

**Returns** Dict describing the new node, containing the ‘name’, ‘level’, and ‘path’ and optionally ‘dependency_dict’ and/or ‘omsi_file_dependencydata’ and any additional data generated by the metadata_generator function
add_dependency(dependency_data)
Add a new dependency to the collection.

Parameters dependency_data (omsi.shared.omsi_dependency_data) – The analysis dependency specification.

Returns the newly created omsi_file_dependencydata object

Raises KeyError in case that a dependency with the same name already exists

get_all_dependency_data(omsi_dependency_format=True)
Get all direct dependencies associated with the analysis.

Parameters omsi_dependency_format – Should the dependencies be retrieved as omsi_analysis_dependency object (True) or as an omsi_file_dependencydata object (False).

Returns List dependency_dict objects containing either omsi file API objects or h5py objects for the dependencies. Access using [index]['name'] and [index]['data'].

get_all_dependency_data_graph(include_omsi_dependency=False, include_omsi_file_dependencydata=False, recursive=True, level=0, name_key='name', prev_nodes=None, prev_links=None, parent_index=None, metadata_generator=None, metadata_generator_kwargs=None)
Get all direct and indirect dependencies associated with the analysis in form of a graph describing all nodes and links in the provenance hierarchy.

Parameters

• include_omsi_dependency – Should the dependency_dict object be included in the entries in the nodes dict?

• include_omsi_file_dependencydata – Should the omsi_file_dependencydata object be included in the entries in the nodes dict?

• recursive – Should we trace dependencies recursively to construct the full graph, or only the direct dependencies. Default true (ie., trace recursively)

• name_key – Which key should be used in the dicts to indicate the name of the object? Default value is ‘name’

• level – Integer used to indicated the recursion level. Default value is 0.

• prev_nodes – List of nodes that have been previously generated. Note, this list will be modified by the call. Note, each node is represented by a dict which is expected to have at least the following keys defined, path, name_key, level (name_key refers to the key defined by the input parameter name_key).

• prev_links – Previously established links in the list of nodes. Note, this list will be modified by the call.

• parent_index – Index of the parent node in the list of prev_nodes for this call. May be None in case the parent we are calling this function for is not yet in the list. If None, then we will add our own parent that contains the dependencies to the list.

• metadata_generator – Optional parameter. Pass in a function that generates additional metadata about a given omsi API object. Note, the key’s level and path and name (i.e., name_key) are already set by this function. The metadata_generator may overwrite these key’s, however, the path has to be unique as it is used to identify duplicate nodes. Overwriting the path with a non-unique value, hence, will lead to errors (missing entries) when generating the graph. Note, the metadata_generator function needs to support the following keyword arguments:
- in_dict: The dictionary to which the metadata should be added to.
- obj: The omsi file API object for which metadata should be generated
- name: A qualifying name for the object
- name_key: The key to be used for storing the name

**metadata_generator_kwargs** – Dictionary of additional keyword arguments that should be passed to the metadata_generator function.

**Returns**

Dictionary containing two lists. 1) nodes [List of dictionaries, describing the elements] in the dependency graph. 2) links: List of tuples with the links in the graph. Each tuple consists of two integer indices for the nodes list. For each node the following entries are given:

- dependency_dict: Optional key used to store the corresponding dependency_dict object. Used only of include_omsi_dependency is True.
- omsi_file_dependencydata: Optional key used to store the corresponding omsi_file_dependencydata object. Used only of include_omsi_file_dependencydata is True.
- name: Name of the dependency. The actual key is specified by name_key
- level: The recursion level at which the object occurs.
- path: The full path to the object
- filename: The full path to the file
- ...: Any other key/value pairs from the dependency_dict dict.

**get_all_dependency_data_recursive** (omsi_dependency_format=True,
                                        omsi_main_parent=None, dependency_list=None)

Get all direct and indirect dependencies associated with the analysis.

**NOTE:** omsi_main_parent and omsi_main_parent are used primarily to ensure that the case of circular dependencies are supported properly. Circular dependencies may occur in the case of semantic dependencies (rather than pure use dependencies), e.g., two datasets that are related modalities may reference each other, e.g., MS1 pointing to related MS2 data and the MS2 datasets referencing the corresponding MS1 datasets.

**Parameters**

- omsi_dependency_format – Should the dependencies be retrieved as dependency_dict object (True) or as an omsi_file_dependencydata object (False)
- omsi_main_parent – The main parent for which the dependencies are calculated. This is needed to avoid recursion back into the main parent for which we are computing dependencies and avoiding that it is added itself as a dependency for itself. If set to None, then we will use the omsi_object associated with the parent group of the dependency group.
- dependency_list – List of previously visited/created dependencies. This is needed only to avoid deep recursion and duplication due to circular dependencies

**Returns** List analysis_data objects containing either omsi file API interface objects or h5py objects for the dependencies. Access using [index]['name'] and [index]['data'].

**get_dependency_omsiobject** (name, recursive=True)
Get the omsi file API object corresponding to the object the dependency is pointing to.

**Parameters**

- **name** – Name of the dependency object to be loaded.
- **recursive** – Should the dependency be resolved recursively, i.e., if the dependency points to another dependencies. Default=True.

**Returns** An omsi file API object (e.g., omsi_file_analysis or omsi_file_msidata) if the link points to a group or the h5py.Dataset the link is pointing to.

`get_omsi_file_dependencydata(name)`

Retrieve the omsi_file_dependencydata object for the dependency with the given name.

**class** `omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata(dependency_group)`

**Bases:** `omsi.dataformat.omsi_file.common.omsi_file_common`

Class for managing data groups used for storing data dependencies.

Create a new omsi_file_dependencydata object for the given h5py.Group

**Parameters** **dependency_group** (h5py.Group with a corresponding omsi type) – h5py.Group object with the dependency data

`get_dataset_name()`

Get the string indicating the name of dataset. This may be empty as it is only used if the dependency points to an object within a managed omsi API object.

**Returns** String indicating the name of the optional dataset.

`get_dependency_objecttype(recursive=True)`

Indicated the type of the object the dependency is pointing to.

**Parameters** **recursive** – Should dependencies be resolved recursively, i.e., if the dependency points to another dependencies. Default=True.

**Returns** String indicating the class of the omsi file API class that is suited to manage the dependency link or the name of the corresponding h5py class.

`get_dependency_omsiobject(recursive=True, external_mode=None)`

Get the omsi file API object corresponding to the object the dependency is pointing to.

**Parameters**

- **recursive** – Should dependencies be resolved recursively, i.e., if the dependency points to another dependencies. Default=True.
- **external_mode** – The file open mode (e.g., ‘r’, ‘a’) to be used when we encounter external dependencies, i.e., dependencies that are stored in external files. By default this is set to None, indicating that the same mode should be used in which this (i.e., the current file describing the dependency) was opened. Allowed modes are ’r’, ’r+’, and ’a’. The modes ’w’, ’w+’, ’x’ are prohibited to ensure that we do not break external files.

**Returns** An omsi file API object (e.g., omsi_file_analysis or omsi_file_msidata) if the link points to a group or the h5py.Dataset the link is pointing to.

`get_dependency_type()`

Get the string describing the type of the dependency

**NOTE:** If the type is missing in the file but we have a parameter name specified, then the default type ‘parameter’ will be returned other None is returned.

**Returns** String indicating the type of the dependency or None if the type is not known.
get_link_name()
Get the name of the dependency link

Returns  String indicating the name of the dependency link.

get_mainname()
Get the main name string describing the name of the object (and possibly path of the file if external)

Returns  String indicating the main name of the object that we link to

get_omsi_dependency()
Get the dependency information as an omsi.shared.dependency_dict object (as defined in the omsi.shared.dependency_dict module)

Returns  dependency_dict object with all the dependency data.

getParameterHelp()
Get the help string for the parameter name if available.

getParameterName()
Get the string indicating the name of the dependend parameter of the analysis.

Returns  String of the parameter name that has the dependency.

getSelectionString()
String indicating the applied selection. This is an empty string in case no selection was applied.

Returns  Selection string. See the omsi.shared.omsi_data_selection for helper functions to deal with selection strings.

file_reader_base Module
Module for base classes for implementation and integration of third-party file readers.

ToDo:
• get_number_of_regions(...) should be updated to return a list of regions, one per dataset
• Need to add base class for multi dataset formats
• Need to add base class for multi dataset+region formats
• Need to implement new file format for combined raw data file (ie., multiple raw files in one folder).

class omsi.dataformat.file_reader_base.file_reader_base(basename, requires_slicing=True)

Bases: object

Base-class used to define the basic interface that file-readers for a new format need to implement.

__init__ interface:
To avoid the need for custom code subclasses should be able to be constructed by providing just the basename parameter and optional requires_slicing parameter. If additional inputs are needed, then file conversion and management scripts may need to be modified to account for the custom requirements. Required Attributes :

Variables
• data_type  – String indicating the data type to be used (e.g., uint16)
• shape  – Tuple indicating the shape of the data
• mz  – Numpy array with the m/z axis data. In the case of multi-data this is a list of numpy arrays, one per dataset.
• **basename** – The basename provided for opening the file.

### Required Interface Functions:

- **close_file**: Close any opened files
- **is_valid_dataset**: Check whether a given dir/file is valid under the current format
- **spectrum_iter**: Generator function that iterates over all the spectra in the current data cube and yield the numpy array with the intensity and integer x, y position of the spectrum

### Optional Interface Functions:

- **__getitem__** [Implement array slicing for files. Required if the requires_slicing parameter] should be supported.
- **supports_regions**: Specify whether the format supports multiple regions (default=False)
- **supports_multidata**: Specify whether the format supports multiple datasets (default=False)
- **supports_multiexperiment**: Specify whether the format supports multiple experiments (default=False)

Construct the base class and define required attributes.

**Parameters**

- **basename** – The name of the file or directory with the file to be opened by the reader
- **requires_slicing** – Boolean indicating whether the user requires array slicing via the __getitem__ function to work or not. This is an optimization, because many MSI data formats do not easily support arbitrary slicing of data but rather only iteration over spectra.

#### static available_formats()

Get dictionary of all available file formats that implement the file_format_base API.

**Returns** Dictionary where the keys are the names of the formats and the values are the corresponding file reader classes.

#### close_file()

Close the file.

#### classmethod format_name()

Define the name of the format.

**Returns** String indicating the name of the format.

#### get_dataset_metadata()

Get dict of additional metadata associated with the current dataset

**NOTE**: In the case that multiple regions and/or datasets are supported, this function should return the metadata of the currently selected dataset only. If no particular dataset is selected, then all should be returned.

**Returns** Instance of omsi.shared.metadata_data.metadata_dict

#### get_number_of_datasets()

File readers with multi dataset support must overwrite this function to retrieve the true number of raw datasets in the file. Default implementation returns 1.

#### get_number_of_regions()

File readers with multi region support must overwrite this function to retrieve the true number of regions in the file. Default implementation returns 1.
**BASTet: Berkeley Analysis and Storage Toolkit, Release devel**

**classmethod is_valid_dataset (name)**  
Classmethod used to check whether a given directory (or file) defines as valid data file of the file format specified by the current child class  

**Parameters** name (String) – Name of the dir or file.  
**Returns** Boolean indicating whether the given file or folder is a valid file.

**classmethod size (name)**  
Classmethod used to check the estimated size for the given file/folder.  

**Parameters** name (String) – Name of the dir or file.  
**Returns** Integer indicating the size in byte or None if unknown.

**spectrum_iter ()**  
Enable iteration of the spectra of the current data cube.  

Iterate over all the spectra in the current data cube and yield the numpy array with the intensity and integer x, y position of the spectrum.  

**NOTE:** As this is a generator one needs to use yield.  
**Returns**  
The function yields for each spectrum the following information:  
• tuple of (x,y) or (x,y,z) position of the spectrum  
• Numpy array with the spectrum

**classmethod supports_multidata ()**  
Define whether the file format support multiple independent datasets.

**classmethod supports_multiexperiment ()**  
Define whether the file format supports multiple independent experiments, each of which may contain multiple datasets.

**classmethod supports_regions ()**  
Define whether the file format support multiple regions.

**class omsi.dataformat.file_reader_base.file_reader_base_multidata (basename, requires_slicing)**

**Bases:** omsi.dataformat.file_reader_base.file_reader_base

Base-class used to define the basic interface for file-readers used to implement new file formats with support for multiple dataset (e.g, MSI dataset with multiple spectrum types). This class extends file_reader_base, and accordingly all required attributes and functions of file_reader_base must be implemented by subclasses.

In addition to the file_reader_base functions we need to implement the get_number_of_datasets(...) and get_dataset_dependencies(...) functions.

**Variables** select_dataset – Unsigned integer indicating the currently selected dataset

Construct the base class and define required attributes.

**get_dataset_dependencies ()**  
Get the dependencies between the current dataset and any of the other datasets stored in the current file. If self.select_dataset is not set, then the function is expected to return a list of lists with all dependencies for all datasets.

**Returns**  
List of dependencies (or list of lists of dependencies if self.select_dataset is None) where each dependency is a dict of the following form:
• 'omsi_object': None, # The omsi file API object where the data is stored. Often None.
• 'link_name': ms2_link_name, # Name for the dependency link to be used
• 'basename': basename, # Basename of the file
• 'region': None, # Index of the region in the dataset or None
• 'dataset': ind2, # Index of the dataset within the file or None
• 'help': scan_types[ms1scan], # Help describing the dependency
• 'dependency_type': ... } # Type of dependency see dependency_dict.dependency_type for available types

get_number_of_datasets()
Get the number of available datasets.

set_dataset_selection(dataset_index)
Define the current dataset to be read.

classmethod supports_multidata()
Define whether the file format supports multiple data blocks.

class omsi.dataformat.file_reader_base.file_reader_base_with_regions(basename, requires_slicing)

Bases: omsi.dataformat.file_reader_base.file_reader_base

Base-class used to define the basic interface for file-readers used to implement new file formats with support for multiple imaging regions per file. This class extends file_reader_base, and accordingly all required attributes and functions of file_reader_base must be implemented by subclasses.

Additional required attributes:
• select_region: Integer indicating which region should be selected. If set to None, indicates that the data should be treated as a whole. If set to a region index, then the data should be treated by the reader as if it only pertains to that region, i.e., the shape of the data should be set accordingly and __getitem__ should behave as such as well.
• region_dicts: List of dictionaries, where each dictionary describes a given region (e.g., the origin and extend for rectangular regions.

Construct the base class and define required attributes.

get_dataset_dependencies()
Get the dependencies between the current region and any of the other region datasets stored in the current file. If self.select_region is not set, then the function is expected to return a list of lists with all dependencies for all datasets.

Returns
List of dependencies (or list of lists of dependencies if self.select_dataset is None) where each dependency is a dict of the following form:
• 'omsi_object': None, # The omsi file API object where the data is stored. Often None.
• 'link_name': ms2_link_name, # Name for the dependency link to be used
• 'basename': basename, # Basename of the file
• 'region': None, # Index of the region in the dataset or None
• 'dataset': ind2, # Index of the dataset within the file or None
• 'help': scan_types[ms1scan], # Help describing the dependency
• dependency_type’: ... } # Type of dependency see dependency_dict.dependency_type for available types

get_number_of_regions()
Get the number of available regions

get_region_selection()
Get the index of the selected region

get_regions()
Get list of all region dictionaries defining for each region the origin and extend of the region. See also self.region_dicts.

set_region_selection(region_index=None)
Define which region should be selected for local data reads.

Parameters
region_index – The index of the region that should be read. The shape of the data will be adjusted accordingly. Set to None to select all regions and treat the data as a single full 3D image.

classmethod supports_regions()
Define whether the file format support multiple regions.

img_file Module

This module provides functionality for reading img mass spectrometry image files

class omsi.dataformat.img_file.img_file (hdr_filename=None, t2m_filename=None, img_filename=None, basename=None, requires_slicing=True)
Bases: omsi.dataformat.file_reader_base.File_reader_base

Interface for reading a single 2D img file

The img format consists of three different files: i) hdr header file, ii) t2m which contains the m/z data, iii) img data file.

Open an img file for data reading.

Parameters
• hdr_filename (string) – The name of the hdr header file
• t2m_filename (string) – The name of the t2m_filename
• img_filename (string) – The name of the img data file
• basename (string) – Instead of img_filename, t2m_filename, and hdr_filename one may also supply just a single basename. The basename is completed with the .img, .t2m, .hdr extension to load the data.
• requires_slicing (Boolean) – Unused here. Slicing is always supported by this reader.

Raises ValueError In case that basename and hdr_filename, t2m_filename, and img_filename are specified.

close_file ()
Close the img file

classmethod get_files_from_dir (dirname)
Get a list of all basenames of all img files in a given directory. Note: The basenames include the dirname.
**is_valid_dataset** *(name)*
Check whether the given file or directory points to a img file.

*Parameters* name *(unicode)* – Name of the dir or file.

*Returns* Boolean indicating whether the given file or folder is a valid img file.

**size** *(name)*
Classmethod used to check the estimated size for the given file/folder.

*Parameters* name *(unicode)* – Name of the dir or file.

*Returns* Integer indicating the size in byte or None if unknown.

**spectrum_iter()**
Enable iteration over the spectra in the file

*Returns* tuple of ((x, y), intensities), i.e., the tuple of (x, y) integer index of the spectrum and the nzym array of the intensities

**mzml_file Module**

This module provides functionality for reading mzml mass spectrometry image files.

```python
filename = '/Users/oruebel/Devel/openmsi-data/mzML_Data/N2A2_Serratia_spots_extract_TI.mzML'
```

**mzml_file(basename, requires_slicing=True, resolution=5)**

Bases: omsi.dataformat.file_reader_base.file_reader_base_multidata

Class for reading a single 2D mzml file with several distinct scan types.

*Variables* available_mzml_types – Dict of available mzml flavors.

Open an img file for data reading.

*Parameters*

- **basename** *(string)* – The name of the mzml file. If basename is a directory, then the first mzML file found in the directory will be used instead.
- **requires_slicing** *(bool)* – Should the complete data be read into memory (this makes slicing easier). (default is True)
- **resolution** *(float)* – For profile data only, the minimum m/z spacing to use for creating the “full” reprofiled data cube

```python
_mzml_file__compute_coordinates()()
```

Internal helper function used to compute the coordinates for each scan.

*Returns* 2D numpy integer array of shape (numScans,2) indicating for each scan its x and y coordinate

**mzml_file Compute Filetype** *(filename)*

Internal helper function used to compute the filetype.

**mzml_file Compute Mz Axis** *(filename, mzml_filetype, scan_types, resolution)*

Internal helper function used to compute the mz axis of each scantype Returns a list of numpy arrays

**mzml_file Compute Num Scans** *(filename=None)*

Internal helper function used to compute the number of scans in the mzml file.

**mzml_file Compute Scan Dependencies** *(scan_types=None, basename=None)*

Takes a scan_types list and returns a list of tuples (x, y) indicating that scan_type[y] depends on scan_type[x]
Internal helper function used to compute a list of unique scan types in the mzml file. Also computes a numpy 1d array of ints which index every scan to relevant datacube.

Internal helper function used to parse out scan parameters from the scan filter string

Internal helper function used to read all data. The function directly modifies the self.data entry. Data is now a list of datacubes

available_mzml_types = {'unknown': 'unknown', 'bruker': 'bruker', 'thermo': 'thermo'}

Close the mzml file

Get the dependencies between the current dataset and any of the other datasets stored in the current file.

Get dict of additional metadata associated with the current dataset.

Returns Dict where keys are strings and associated values to be stored as metadata with the dataset.

Get a list of all basenames of all img files in a given directory. Note: The basenames include the dirname.

Get the number of available datasets.

Check whether the given file or directory points to a img file.

Parameters name (String) – Name of the dir or file.

Returns Boolean indicating whether the given file or folder is a valid img file.

Define the current dataset to be read.

Classmethod used to check the estimated size for the given file/folder. For mzml this is an estimate of the final size of the full 3D datacube. For efficiency the number of scans is estimated based on the size of the first 1000 scans.

Parameters

• name (unicode) – Name of the dir or file.

• max_num_reads (int) – The maximum number of spectrum reads to be performed to estimate the file size

Returns Integer indicating the size in byte or None if unknown.

Generator function that yields a position and associated spectrum for a selected datacube type.

Yield (xidx, yidx) a tuple of ints representing x and y position in the image
Yield  $y_i$, a numpy 1D-array of floats containing spectral intensities at the given position and for
the selected datacube type

```python
classmethod test()
    Test method
```

**bruckerflex_file Module**

This module provides functionality for reading bruker flex mass spectrometry image files

Limitations:

1. Currently the reader assumes a single global m/z axis for all spectra.
2. The read of acqu files does not convert \(<...>\) entries to python values but leaves them as strings.
3. \_read\_spotlist\__ converts the regions to start with a 0 index. This is somewhat inconsistent in the spot list file.
   The spotname seems to number regions starting with 0 while the region list numbers them starting with 1.
4. \_read\_spotlist\__ computes the folder where the spots are located based on the filename of the spotlist. The
   question is whether this is always the case?? The advantage is that we do not rely on the regions.xml file which
   contains absolute paths which are in most cases invalid as the data has been copied between different machines
   in many cases and is stored in different locations on each of the machines.
5. \_read\_spotlist\__ currently assumes that there is only one fid file per spot
6. \_read\_spotlist\__ currently only looks at where the acqu and fid file is located for the spot. Other files are
   currently ignored.
7. \_read\_spotlist\__ (and hence the reader at large) currently assumes that we have 2D images only.
8. \_read\_spotlist\__ currently generates maps for the image that assume that x and y pixel indices start at 0. Not
   all images may record data until the border, so that this strategy may add empty spectra rather than generating a
   new bounding box for the image.
9. \_read\_spotlist\__ assumes in the variable spotname\_encoding a maximum of 24 characters in the spotname
   R01X080Y013. This should in general be more than sufficient as this allows for 7 characters for each R, X, Y
   entry, however, if this is not enough then this behaviour needs ot be changed.
10. \_getitem\__ currently only works if we have read the full data into memory. An on-demand load should be
    supported as well.
11. We can currently only selected either a single region or the full data but we cannot selected multiple regions at
    once. E.g. if a dataset contains 3 regions then we can either select all regions at once or region 1,2, or 3 but one
    cannot selected region 1+2, 1+3, or 2+3.

```python
import bruckerflex_file
spotlist = "/Users/oruebel/Devel/msidata/Bruker_Data/UNC IMS Data/20130417 Bmy

coides Paenibacillus Early SN03130/" + "2013 Bmyc Paeni Early LP/2013 Bmyc Paeni Early LP Spot List.txt"
exppath ="/Users/oruebel/Devel/msidata/Bruker_Data/UNC IMS Data/20130417 Bmycoides Paenibacillus Early
SN03130/" + "2013 Bmyc Paeni Early LP/2013 Bmyc Paeni Early LP/0_R00X012Y006/1/1SLin" f = bruck-
erflex_file.bruckerflex_file( spotlist_filename = spotlist) f.s_read_fid( exppath+"/fid" , f.data_type ) testacqu =
f.s_read_acqu( exppath+"/acqu" ) testmz = f.s_mz_from_acqu( testacqu ) testspotlist = f.s_read_spotlist(spotlist)
a = bruckerflex_file( dirname )

class omsi.dataformat.bruckerflex_file.bruckerflex_file(basename,
    fid_encoding='int32',
    requires_slicing=True)
Bases: omsi.dataformat.file_reader_base.file_reader_base_with_regions
    Interface for reading a single bruker flex image file.
```

172  Chapter 10. omsi Package
The reader supports standard array slicing for data read. I.e., to read a spectrum use [x,y,:] to read an ion image using [:,:,:].

The reader supports multiple regions, i.e., reading of different independent regions that were imaged as part of the same dataset. Using the get_regions, get_number_of_regions, set_region_selection and get_region_selection the user can interact with the region settings. Using the set_region_selection the user can define whether the data of the complete image should be read (set_region(None), default) or whether data from a single region should be read. When a region is selected, then the reader acts as if the region were the complete dataset, i.e., the shape variable is adjusted to fit the selected region and the __get_item__ method (which is used to implement array-like slicing (e.g., [1,1,:])) behaves as if the selected region where the full data.

Open an img file for data reading.

**Parameters**

- **basename (string)** – Name of the textfile with the spotlist. Alternatively this may also be the folder with the spots.
- **requires_slicing (bool)** – Should the complete data be read into memory (this makes slicing easier). (default is True)
- **fid_encoding (string)** – String indicating in which binary format the intensity values (fid files) are stored. (default value is ‘int32’)

**Variables**

- **self.basename** – Name of the file with the spotlist
- **self.pixel_dict** – dictionary with the pixel array metadata (see also s_read_spotlist(...)). Some of the main keys of the dictionary are, e.g. (see also s_read_spotlist(...)):
  - ‘spotfolder’ : String indicating the folder where all the spot-data is located
  - ‘fid’ : 2D numpy masked array of strings indicating for (x,y) pixel the fid file with the intensity values.
  - ‘acqu’ : 2D numpy masked array of strings indicating for each (x,y) pixel the acqu file with the metadata for that pixel.
  - ‘regions’ : 2D numpy masked array of integers indicating for each pixels the index of the region it belongs to.
  - ‘xpos’ : 2D numpy masked array of integers indicated for each pixel its x position.
  - ‘ypos’ : 2D numpy masked array of integers indicated for each pixel its x position.
  - ‘spotname’ : 2D masked numpy array of strings with the names of the spot corresponding to a pixel.
- **self.data_type** – the encoding used for intensity values
- **self.shape** – The 3D shape of the MSI data volume for the currently selected region.
- **self.full_shape** – Shape of the full 3D MSI dataset including all regions imaged.
- **self.metadata** – Dictionary with metadata from the acqu file
- **self.mz** – The 1D numpy array with the m/z axis information
- **self.data** – If requires_slicing is set to true then this 3D array includes the complete data of the MSI data cube. Missing data values (e.g., from regions not imaged during the acquisition processes) are completed with zero values.
- **self.region_dicts** – Dictionary with description of the imaging regions
Raises ValueError  In case that no valid data is found.

close_file()
Close the img file

get_dataset_dependencies()
Get the dependencies between the current region and any of the other region datasets stored in the current file. If self.select_region is not set, then the function is expected to return a list of lists with all dependencies for all datasets.

Returns
List of dependencies (or list of lists of dependencies if self.select_dataset is None) where each dependency is a dict of the following form:
- 'omsi_object': None, # The omsi file API object where the data is stored. Often None.
- 'link_name': ms2_link_name, # Name for the dependency link to be used
- 'basename': basename, # Basename of the file
- 'region': None, # Index of the region in the dataset or None
- 'dataset': ind2, # Index of the dataset withing the file or None
- 'help':scan_types[ms1scan], # Help describing the dependency
- 'dependency_type': ... } # Type of dependency see dependency_dict.dependency_type for available types

classmethod is_valid_dataset(name)
Determine whether the given file or name specifies a bruckerflex file

Parameters name (string) – name of the file or dir

Returns Boolean indicating whether the name is a valid bruckerflex

static s_mz_from_acqu(acqu_dict)
Construct the m/z axis from the data stored in the acqu_dict dictionary. See also s_read_acqu

Parameters acqu_dict – Python dictionary with the complete information from the acqu file.
See s_read_acqu(...).

returns 1D Numpy array of floats with the mz axis data.

static s_read_acqu(filename)
Construct an m/z axis for the given acqu file.

Parameters filename (string) – String with the name+path for the acqu file.

Returns Return dictionary with the parsed metadata information

static s_read_fid(filename, data_type='int32', selection=slice(None, None))
Read data from an fid file

Parameters
- filename (string) – String indicating the name+path to the fid file.
- data_type – The numpy datatype encoding used for the fid files. (default is ‘int32’). In the instance of this class this is encoded in the data_type variable associated with the instance.
- selection (slice or list, i.e., a selection that numpy understands) – This may be a python slice or a list of indecies to be read. Default value is to read all (i.e., slice(0,None,None))
Returns 1D numpy array of intensity values read from the file.

**static s_read_spotlist (spotlist_filename)**

Parse the given spotlist file.

**Parameters**

*spotlist_filename* (*string*) – Name of the textfile with the spotlist

**Returns**

The function returns a number of different items in from of a python dictionary. Most data is stored as 2D spatial maps, indicating for each (x,y) location the corresponding data. Most data is stored as 2D masked numpy arrays. The masked of the array indicated whether data has been recorded for a given pixel or not. The dict contains the following keys:

- ‘spotfolder’ : String indicating the folder where all the spot-data is located
- ‘fid’ : 2D numpy masked array of strings with fid file name for each (x,y) pixel (intensities).
- ‘acqu’ : 2D numpy masked array of strings with acqu file name for each (x,y) (metadata).
- ‘regions’ : 2D numpy masked array of integers with region index for each (x,y) pixel.
- ‘xpos’ : 2D numpy masked array of integers indicated for each pixel its x position.
- ‘ypos’ : 2D numpy masked array of integers indicated for each pixel its y position.
- ‘spotname’ : 2D masked numpy array of strings with spot name for each (x,y) pixel.

**static s_spot_from_dir (in_dir, spot_folder_only=False)**

Similar to s_read_spotlist but instead of using a spotlist file the structure of the data is parsed directly from the structure of the directory containing all spots.

**param in_dir** Name of the directory with all spots

**type in_dir** string

**param spot_folder_only** If set to True, then the function only constructs the spot folders but does not check for acqu files etc. If set to True, only the spotfolder list will be returned.

**returns**

The function returns None in case that no valid spots were found. Returns a list of strings with the spotfolders if spot_folder_only is set to True. Otherwise, the function returns a number of different items in from of a python dictionary. Most data is stored as 2D spatial maps, indicating for each (x,y) location the corresponding data. Most data is stored as 2D masked numpy arrays. The masked of the array indicated whether data has been recorded for a given pixel or not. The dict contains the following keys:

- ‘spotfolder’ : String indicating the folder where all the spot-data is located
- ‘fid’ : 2D numpy masked array of strings with fid file name for each (x,y) pixel (intensities)
- ‘acqu’ : 2D numpy masked array of strings with acqu file name for each (x,y) (metadata)
- ‘regions’ : 2D numpy masked array of integers with the index of the region for each (x,y) pixel.
• ‘xpos’: 2D numpy masked array of integers indicated for each pixel its x position.
• ‘ypos’: 2D numpy masked array of integers indicated for each pixel its x position.
• ‘spotname’: 2D masked numpy array of strings with the name of the spot corresponding to a pixel.

`set_region_selection(region_index=None)`
Define which region should be selected for local data reads.

Parameters
`region_index` – The index of the region that should be read. The shape of the data will be adjusted accordingly. Set to None to select all regions and treat the data as a single full 3D image.

## 10.1.3 datastructures Package

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>omsi.datastructures</code></td>
<td>Package with a collection of various data structures and related classes used throughout the software stack, e.g., for metadata, analysis parameter data, runtime information data etc.</td>
</tr>
<tr>
<td><code>omsi.datastructures.metadata</code></td>
<td>Package with metadata datastructures</td>
</tr>
<tr>
<td><code>omsi.datastructures.metadata.metadata_data</code></td>
<td>Define infrastructure for describing metadata (in memory)</td>
</tr>
<tr>
<td><code>omsi.datastructures.metadata.metadata_ontologies</code></td>
<td>Define ontologies for metadata</td>
</tr>
<tr>
<td><code>omsi.datastructures.analysis_data</code></td>
<td>Helper module with data structures for managing analysis-related data.</td>
</tr>
<tr>
<td><code>omsi.datastructures.dependency_data</code></td>
<td>Define a dependency to another omsi object</td>
</tr>
<tr>
<td><code>omsi.datastructures.run_info_data</code></td>
<td>Module with helper data structures for recording runtime provenance data.</td>
</tr>
</tbody>
</table>

### datastructures Package

Package with a collection of various data structures and related classes used throughout the software stack, e.g., for metadata, analysis parameter data, runtime information data etc.

### analysis_data Module

Helper module with data structures for managing analysis-related data.

```python
class omsi.datastructures.analysis_data.analysis_data(name='undefined', data=None, dtype='float32')
```

Bases: `dict`

Define an output dataset for the analysis that should be written to the omsi HDF5 file

The class can be used like a dictionary but restricts the set of keys that can be used to the following required keys which should be provided during initialization.

**Required Keyword Arguments:**

Parameters

- `name` – The name for the dataset in the HDF5 format
- `data` – The numpy array to be written to HDF5. The data write function `omsi_file_experiment.create_analysis` used for writing of the data to file can in principal also handle other primitive data types by explicitly converting them to numpy. However, in this case the dtype is determined based on the numpy conversion and correct behavior is not guaranteed. I.e., even single scalars should be stored as a 1D numpy array here. Default value is None which is mapped to np.empty( shape=(0), dtype=dtype) in `__init__`
• **dtype** – The data type to be used during writing. For standard numpy data types this is just the dtype of the dataset, i.e., ['data'].dtype. Other allowed datatypes are:
  - For string: omsi_format.str_type (omsi_format is located in omsi.dataformat.omsi_file)
  - To generate data links: ana_hdf5link (analysis_data)

```python
ana_hdf5link = -1
```

Value used to indicate that a hard link to another dataset should be created when saving an analysis object.

```python
class omsi.datastructures.analysis_data.data_dtypes
Bases: dict

Class specifying basic function for specifying common data types used as part of an analysis.

```python
static bool_type (argument)
```

Implement conversion of boolean input parameters since argparse (or bool, depending on the point of view), do not handle bool as a type in an intuitive fashion.

**Parameters**
- **argument** – The argument to be parsed to a boolean

**Returns**
The converted value

```python
static get_dtypes ()
```

Get a list of available data type specifications

```python
static ndarray (argument)
```

This dtype may be used to indicate numpy ndarrays as well as h5py arrays or omsi_dependencies

**Parameters**
- **argument** – The argument to be parsed to ndarray

**Returns**
The converted ndarray

```python
class omsi.datastructures.analysis_data.parameter_data (name, help='', dtype=None, required=False, default=None, choices=None, data=None, group=None)
```

Bases: dict

Define a single input parameter for an analysis.

**Variables**
- **default_keys** – List of allowed dictionary keys:

Required keys:

- **name** : The name of the parameter
- **help** : Help string describing the parameter
- **type** : Optional type. Default is None, indicating a dynamically typed dataset that the analysis will convert
- **required** : Boolean indicating whether the parameter is required (True) or optional (False). Default False
- **default** : Optional default value for the parameter. Default None.
- **choices** : Optional list of choices with allowed data values. Default None, indicating no choices set.
- **data** : The data assigned to the parameter. None by default.
- **group** : Optional group string used to organize parameters. This may also be a dict of {‘name’:<group>, ‘description’:<description>}

In the context of the argparse package the default keys have the following mapping:

- **argparse.name = name**
- **argparse.action -->** The action is constant and set to save value
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

• `argparse.nargs` -> Left as default
• `argparse.const` -> Not used as action is always save value
• `argparse.type = type`
• `argparse.choices = 'choices`
• `argparse.required = 'required`
• `argparse.help = 'help`
• `argparse.metavar` -> Not used. Positional arguments are not allowed for analyses
• `argparse.destination` -> Automatically determined by the `name` of the parameter
• `argparse.add_argument_group(...)` -> Automatically determined based on the required parameter and the `group` parameter if set.

Initialize a new parameter description.

Parameters

• **name** – Required name for the parameter
• **help** – Required help string for the parameter
• **dtype** – Type argument. Default unicode.
• **required** – Boolean indicating whether the parameter is required (default=True)
• **default** – Optional default value for the parameter. Default None.
• **choices** – Optional list of choices with allowed data values. Default None, indicating no choices set.
• **data** – The data assigned to the parameter. None by default.
• **group** – The parameter group to be used. None by default.

clear_data()

Remove the currently assigned data.

copy()

Return a new parameter_data object with the same data as stored in the current object

Returns dependency_dict object

data_ready()

This function check if the data points to a dependency and if so, then check if the dependency can be resolved or not

data_set()

Check if a data has been assigned for the parameter.

default_keys = ['name', 'default', 'dtype', 'choices', 'required', 'help', 'data', 'group']

List of allowed keys for the parameter dict.

get_data_or_default()

Get the data of the parameter if set, otherwise get the default value if available.

Returns The data to be used for the parameter.

Raises KeyError is raised in case that neither ‘default’ nor ‘data’ are available. This should never be the case if the object was created properly.

get_group_description()

Get the description for the group if available.
Returns  String with the group description or None.

get_group_name ()
Get the name of the group to be used.

Returns  String with the name of the group of None if not set

is_dependency ()
Check whether the parameter defines a dependency.

Returns  Boolean indicating whether the parameter defines a dependency.

class omsi.datastructures.analysis_data.parameter_manager
Bases: object

Base class for objects that manage their own parameters.

Parameters are set and their values retrieved by name using dict-like slicing. Derived classes may overwrite
__getitem__ and __setitem__ to implement their own behavior but we expect that the functionality of the in-
terface is preserved, i.e., others should still be able set parameter value and retrieve values via dict slicing.

add_parameter (name, help, dtype=<type 'unicode'>, required=False, default=None, choices=None,
data=None, group=None)
Add a new parameter for the analysis. This function is typically used in the constructor of a derived
analysis to specify the parameters of the analysis.

Parameters

- name – The name of the parameter

- help – Help string describing the parameter

- dtype – Optional type. Default is string.

- required – Boolean indicating whether the parameter is required (True) or optional
  (False). Default False.

- default – Optional default value for the parameter. Default None.

- choices – Optional list of choices with allowed data values. Default None, indicating
  no choices set.

- data – The data assigned to the parameter. None by default.

- group – Optional group string used to organize parameters. Default None, indicating
  that parameters are automatically organized by driver class (e.g. in required and optional
  parameters)

Raises  ValueError is raised if the parameter with the given name already exists.

clear_parameter_data ()
Clear the list of parameter data

define_missing_parameters ()
Set any required parameters that have not been defined to their respective default values.

This function may be overwritten in child classes to customize the definition of default parameter values
and to apply any modifications (or checks) of parameters before the analysis is executed. Any changes
applied here will be recorded in the parameter of the analysis.

get_all_dependency_data ()
Get the complete list of all direct dependencies to be written to the HDF5 file
NOTE: These are only the direct dependencies as specified by the analysis itself. Use get_all_dependency_data_recursive(..) to also get the indirect dependencies of the analysis due to dependencies of the dependencies themselves.

Returns List of parameter_data objects that define dependencies.

get_all_parameter_data (exclude_dependencies=False)
Get the complete list of all parameter datasets to be written to the HDF5 file

Parameters exclude_dependencies – Boolean indicating whether we should exclude parameters that define dependencies from the list

get_num_dependency_data ()
Return the number of dependencies defined as part of the parameters

get_num_parameter_data ()
Return the number of parameter datasets to be written to the HDF5 file

get_parameter_data (index)
Given the index return the associated dataset to be written to the HDF5 file

:param index [Return the index entry of the private member parameters. If a] string is given, then get_parameter_data_by_name(...) will be used instead.

Raises IndexError is raised when the index is out of bounds

get_parameter_data_by_name (dataname)
Given the key name of the data return the associated parameter_data object.

Parameters dataname – Name of the parameter requested from the parameters member.

Returns The parameter_data object or None if not found

get_parameter_names ()
Get a list of all parameter dataset names (including those that may define dependencies.

keys ()
Get a list of all valid keys, i.e., a list of all parameter names.

Returns List of strings with all input parameter and output names.

set_parameter_default_value (name, value)
Set the default value of the parameter with the given name

Parameters

• name – Name of the parameter

• value – New value

Raises KeyError if parameter not found

dependency_data Module

Define a dependency to another omsi object
class omsi.datastructures.dependency_data.dependency_dict(param_name=None, 
  link_name=None, 
  omsi_object=None, 
  selection=None, 
  dataname=None, 
  help=None, 
  dependency_type=None)

Bases: dict

Define a dependency to another omsi file-based data object or in-memory analysis_base object

Required Keyword Arguments:

Variables

- **param_name** – The name of the parameter that has the depency
- **link_name** – The name of for the link to be created in the HDF5 file.
- **oms_object** – The object to which a link should be established to. This must be either an h5py.Dataset or the omsi_file_analysis or omsi_file_msidata or any of the other omsi_file API interface objects.
- **selection** – Optional string type parameter indicating a python selection for the dependency
- **dataname** – String indicating the dataset within the omsi_object. If the omsi_object is an h5py object within a managed Group, then the omsi_object is automatically split up into the parent object and dataname.
- **_data** – Private key used to store the data associated with the dependency object.

Optional Keyword arguments:

Variables **dependency_type** – The type of the dependency being modeled. If not defined then the default value of ‘parameter’ is assumed.

Initialize the allowed set of keys.

Parameters

- **param_name** – The name of the parameter that has the dependency
- **link_name** – The name of for the link to be created in the HDF5 file.
- **omsi_object** – The object to which a link should be established to. This must be either an h5py.Dataset or the omsi_file_analysis or omsi_file_msidata or any of the other omsi_file API interface objects.
- **selection** – Optional string type parameter indicating a python selection for the dependency
- **dataname** – String indicating the dataset within the omsi_object. If the omsi_object is an h5py object within a managed Group, then the omsi_object is automatically split up into the parent object and dataname.
- **help** – Optional string describing the object

copy()

Return a new dependency_dict object with the same data as stored in the current object

Returns dependency_dict object

dependency_types = {'subset': 'subset', 'undefined': None, 'contains': 'contains', 'link': 'link', 'parameter': 'parameter'}
get_data()
Get the data associated with the dependency.

Returns If a selection is applied and the dependency object supports array data load (e.g., h5py.Dataset, omsi_file_msisdata), then the selected data will be loaded and returned as numpy array. Otherwise the ['omsi_object'] is returned.

run_info_data Module
Module with helper data structures for recording runtime provenance data

class omsi.datastructures.run_info_data.run_info_dict(*args, **kwargs)
Bases: dict
Simple dictionary class for collecting runtime information
The typical use is as follows:
>> my_run_info = run_info_dict() >> my_run_info(my_function)(my_parameters)
With this, all runtime information is automatically collected in my_run_info. We can enable time-and-usage and memory profiling simply by calling enable_profile_time_and_usage(...) or enable_profile_memory(...), respectively, before we run our function.

We can also use the data structure directly and control the population ourselves, however, memory profiling is not supported by default in this case but we need to set and run the memory profiler ourselves, since memory_profiler expects that it can wrap the function

DEFAULT_TIME_FORMAT = '%Y-%m-%d %H:%M:%S.%f'
clean_up()
Clean up the runinfo object. In particular remove empty keys that either recorded None or recorded just an empty string.
This function may be overwritten to also do clean-up needed due to additional custom runtime instrumentation.
When overwriting this function we should call super(..., self).runinfo_clean_up() at the end of the function to ensure that the runinfo dictionary is clean, i.e., does not contain any empty entries.
clear()
Clear the dictionary and other internal parameters
Side Effects
• Remove all key/value pairs from the dict
• Set self.__time_and_use_profiler to None
• Set self.__memory_profiler to None
• Set self.__profile_memory to False if invalid (i.e, if set to True but memory profiling is unavailable)
• Set self.__profile_time_and_usage to False if invalid (i.e., if set to True but profiling is unavailable)

enable_profile_memory(enable=True)
Enable/disable profiling of memory usage
Parameters enable – boolean to enable (True) or disable (False) memory profiling

enable_profile_time_and_usage(enable=True)
Enable/disable time and usage profiling
Parameters enable – boolean to enable (True) or disable (False) time and usage profiling
gather()

Simple helper function to gather the runtime information—that has been collected on multiple processes when running using MPI—on a single root process.

**Returns** If we have more than one processes then this function returns a dictionary with the same keys as usual for the run_info but the values are now lists with one entry per mpi processes. If we only have a single process, then the run_info object will be returned without changes.

NOTE: Similar to mpi gather, the function only collects information on the root. All other processes will return just their own private runtime information.

get_profile_memory()

Check whether profiling of memory usage is enabled.

**Returns** Boolean indicating whether memory profiling is enabled.

get_profile_stats_object (consolidate=True, stream=None)

Based on the execution profile of the execute_analysis(...) function get pstats.Stats object to help with the interpretation of the data.

**Parameters**

- `consolidate` – Boolean flag indicating whether multiple stats (e.g., from multiple cores) should be consolidated into a single stats object. Default is True.
- `stream` – The optional stream parameter to be used for the pstats.Stats object.

**Returns** A single pstats.Stats object if consolidate is True. Otherwise the function returns a list of pstats.Stats objects, one per recorded statistic. None is returned in case that the stats objects cannot be created or no profiling data is available.

get_profile_time_and_usage()

Check whether time and usage profiling is enabled.

**Returns** Boolean indicating whether time and usage profiling is enabled.

record_postexecute (execution_time=None)

Function used to record runtime information after the task we want to track is completed, e.g. the execute_analysis(...) function of a standard analysis.

The function may be overwritten in child classes to add recording of additional runtime information.

When overwriting the function we should call super(...,self).runinfo_record_postexecute(execution_time) in the custom version to ensure that the execution and end_time are properly recorded.

**Parameters**

- `execution_time` – The total time it took to execute the analysis. May be None, in which case the function will attempt to compute the execution time based on the start_time (if available) and the current time.
- `comm` – Used for logging only. The MPI communicator to be used. Default value is None, in which case MPI.COMM_WORLD is used.

record_preexecute()

Record basic runtime information in this dict before the execution is started.

Function used to record runtime information prior to executing the process we want to track, e.g., the execute_analysis(...) of a standard analysis.

The function may be overwritten in child classes to add recording of additional runtime information. All runtime data should be recorded in the main dict (i.e., self). This ensures in the case of standard analysis that the data is stored in the HDF5 file. Other data should be stored in separate variables that we may add to the object.
When overwriting the function we should typically call super(...,self).runinfo_record_pretexecute() last in the custom version to ensure that the start_time is properly recorded right before the execution of the analysis.

```python
static string_to_structime (time_string, time_format=None)
    Convert a time string to a time.struct_time using time.strptime
```

**Parameters**

- **time_string** – String with the time, e.g., with the start time of a program.
- **time_format** – The time format to be used or None in which case run_info_dict.DEFAULT_TIME_FORMAT will be used.

```python
static string_to_time (time_string, time_format=None)
    Convert a time string to local time object using time.mktime.
```

**Parameters**

- **time_string** – String with the time, e.g., with the start time of a program.
- **time_format** – The time format to be used or None in which case run_info_dict.DEFAULT_TIME_FORMAT will be used.

**Subpackages**

**metadata Package**

**metadata Package** Package with metadata datastructures

**metadata_data Module** Define infrastructure for describing metadata (in memory)

```python
class omsi.datastructures.metadata.metadata_data.metadata_data_dict
    Bases: dict
    Dictionary’s for storing metadata information. The values in the dict must be of type omsi_metadata_value and the keys must be strings.

    get_metadata_descriptions ()
    Get a list of all metadata descriptions

    get_metadata_units ()
    Get a list of all metadata units

    get_metadata_values ()
    Get a list of all metadata values.
```

```python
class omsi.datastructures.metadata.metadata_data.metadata_data_value (name, value, description, unit=None, ontology=None)
    Bases: dict
    A single metadata value
```

- **name** The name of the metadata value
- **value** The actual value associated with the metadata object
- **description** The text description of the metadata object
- **unit** The unit string
• ontology Optional ontology

Parameters

• name – The name of the metadata value. Name may be None if the metadata_value is added to a metadata_dict as it will be set (if missing) when adding it to the metadata_dict
• value – The actual value associated with the metadata object
• description – The text description of the metadata object
• unit – The unit string
• ontology – Optional ontology

metadata_ontologies Module Define ontologies for metadata

```python
var METADATA_ONTOLOGIES

Description of simple ontologies for metadata. This is a dict where the values are the descriptions of the ontologies and the key is the commonly used name of the variable associated with the ontology, however, in practice an ontology may be associated with many different metadata values (the purpose of the ontology is to standardize the values not the names of metadata variables). Available ontologies include:

• polarity: Description of the polarity of the instrument
• msn_value_of_n: Numeric level of mass spectrometry used (e.g., 1 for MS1 etc).
```

An ontology can be many things. In general an ontology is a mechanism to formally name and define the types, properties, and interrelationships of entities. We here refer broadly to the concept of ontologies as a means to standardize the names of things. To define an ontology we use simple dicts with the following key/value entries:

• name: The name of the ontology
• value: Python dict describing the actual ontology. Often this is simply a dict where the keys are the allowed values and the value is the text description of the meaning of that value. In many cases this may be a more complete description of an ontology.
• unit: The standard unit associated with the values (or None if not unit is available)
• description: Human-readable textual description of the ontology
• version: The version of the ontology used
• ‘uri’: The Universal Remote Identifier (often a URL) associated with the ontology (or None)

Ontologies are stored in the OpenMSI data format as JSON attributes associated with the metadata, i.e., one main restriction is that ontologies should be JSON serializable (which in most cases should be a problem).

```python
class omsi.datastructures.metadata.metadata_ontologies.metadata_ontologies
    Bases: dict
    Helper class for interacting with ontologies
```

10.1.4 shared Package

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omsi.shared</td>
<td>Package used to implement shared functionality and helper functions.</td>
</tr>
<tr>
<td>omsi.shared.data_selection</td>
<td>Module for defining and processing data selections.</td>
</tr>
<tr>
<td>omsi.shared.log</td>
<td>Module providing functionality for logging based on the python logging module.</td>
</tr>
</tbody>
</table>

10.1. Subpackages
Table 10.12 – continued from previous page

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omsi.shared.mpi_helper</td>
<td>Module used to ease the use of MPI and distributed parallel implementations using MPI.</td>
</tr>
<tr>
<td>omsi.shared.omsi_web_helper</td>
<td>Module with helper functions for interactions with the OpenMSI web infrastructure.</td>
</tr>
<tr>
<td>omsi.shared.spectrum_layout</td>
<td>This module provides capabilities for computing different layouts for spectra.</td>
</tr>
<tr>
<td>omsi.shared.third_party</td>
<td>Package containing shared third-party code modules included here to reduce the need for external dependencies.</td>
</tr>
<tr>
<td>omsi.shared.third_party.cloudpickle</td>
<td>This class is defined to override standard pickle functionality. The goals of it follow:</td>
</tr>
</tbody>
</table>

**data_selection Module**

Module for defining and processing data selections. This includes the definition of selections using strings as well as transformation and reduction of data.

**TODO:** We may want to expose some of the following numpy functions currently not yet supported through the transform and reduce data operations:

- array2string
- array_equal
- array_equiv
- array_repr
- array_split
- array_str
- asanyarray
- asarray
- asarray_chkfinite
- ascontiguousarray
- asfarray
- asfortranarray
- asmatrix
- asscalar
- atleast_1d
- atleast_2d
- atleast_3d
- binary_repr
- convolve
- conjugate
- cross
- dot
- extract
- fft.*
- histogram, histogram2D, histogramdd
- kron
• linalg.*
• swapaxes(a, axis1, axis2)
• transpose

#Simple data transformation and reduction example from omsi.shared.omsi_data_selection import * import numpy as np import json t = [{‘transformation’: ‘threshold’, ‘threshold’:60}, {‘reduction’: ‘max’, ‘axis’:2}] tj = json.dumps(t) a = np.arange(125).reshape((5,5,5)) apro = transform_and_reduce_data(data=a, operations=tj, http_error=True) apro

#Another simple example from omsi.shared.omsi_data_selection import * import numpy as np import json a = np.arange(10)+5 print a # 1) substract minimum # 2) divide by the maximum value with the maximum value converted to float # NOTE: The conversion to float is to avoid division of integers, i.e., # 5/10 = 0, whereas 5/float(10) = 0.5 # NOTE: The specification of ‘x1’: ‘data’ can be omitted as this is the default. # ‘x1’: ‘data’ simply explicitly specifies that the input data should be # assigned to the first operand of the arithmetic operation. t = [{‘transformation’: ‘dualDataTransform’, ‘operation’: ‘subtract’, ‘x1’: ‘data’, ‘x2’: [{‘reduction’: ‘min’}]},

{‘transformation’: ‘dualDataTransform’, ‘operation’: ‘divide’, ‘x1’: ‘data’, ‘x2’: [{‘reduction’: ‘max’}],
 {‘transformation’: ‘astype’, ‘dtype’: ‘float’}]}
b = transform_and_reduce_data(data=a, operations=t) print b t = [{‘transformation’: ‘threshold’, ‘threshold’: [{‘reduction’: ‘median’}]}, print t c = transform_and_reduce_data(data=a, operations=t) print c

Construct a JSON description of a transformation/reduction from omsi.shared.omsi_data_selection import * #Construct the different pieces of the transformation and reduction pipeline #1) Compute the maximum data value and convert it to float #1.1) Compute the maximum value max_value = construct_reduce_dict( reduction_type=’max’, axis=None) #1.2) Convert data to float value_as_float = construct_transform_dict( trans_type=’astype’, dtype=’float’) #1.3) Merge the two steps to compute the maximum data value as float max_value_as_float = construct_transform_reduce_list( max_value, value_as_float) #2) Normalize the data by dividing by the maximum value divide_by_max_value = construct_transform_dict( trans_type=’dualDataTransform’, operation=’divide’, axes=None, x2=max_value_as_float)

#3) Project along the last axis (i.e., the mz axis) to compute a maximum project image max_projection = construct_reduce_dict( reduction_type=’max’, axis=-1) #4) Merge the different steps and construct the json string json_string = transform_reduce_description_to_json( divide_by_max_value, max_projection) #Just copy the result of the following print statement as your JSON description print json_string

omsi.shared.data_selection.check_selection_string(selection_string)
Check whether the given selection string is valid, and indicate which type of selection the string defined. Checking the selection string is meant as a safeguard to prevent attackers from being able to insert malicious code.

Parameters selection_string – String given by the user with the desired selection

Returns  
String indicating the type of selection as defined in selection_type:
• ‘indexlist’ : Selection of the form [1,2,3]
• ‘all’ : Selection of the form ‘:
• ‘range’ : Selection of the form ‘a:b’
• ‘index’ : A single index selection, e.g., ‘1’
• ‘invalid’ : An unsupported selection

omsi.shared.data_selection.construct_reduce_dict(reduction_type, **kwargs)
Helper function used to construct reduction dictionary.

Required Keyword arguments:

Parameters reduction_type – The reduction type to be used.
Optional Keyword arguments:

Parameters

- **axis** – Some reduction functions support the axis parameters, describing along which axis the reduction should be performed.

- **x1** – By default the reductions are performed on the output of the previous data operation (x1='data'). We may reference the output of, e.g., the fifth data operation by setting x1='data5'. x1 itself may also specify a separate data transformation and reduction pipeline that operates on ‘data’.

- **min_dim** – Minimum number of dimensions the input data should have in order for the reduction should be applied.

Returns Dictionary with the description of the reduction operation.

oms/shared/data_selection.construct_transform_dict(trans_type, axes=None, **kwargs)

Helper function used to construct a dictionary describing a data transformation.

Parameters

- **trans_type** – The transformation type to be used. See transformation_type dict.

- **axes** – The axes along which the data should be split. Default is None.

- **kwargs** – Additional keyword parameters for the transformation functions.

Returns Dictionary with the description of the transformation.

Raises KeyError is raised in case that a parameter is missing. ValueError is raised in case that a given parameter value is invalid.

oms/shared/data_selection.construct_transform_reduce_list(*args)

Merge a series of transformations and reductions into a single list describing a pipeline of transformation and reduction operations to be performed.

Args Ordered series of dictionaries describing transformation and reduction operations.

Returns List of all transformation and reduction operations

oms/shared/data_selection.evaluate_transform_parameter(parameter, data=None, secondary_data=None)

Evaluate the given query parameter. This function is used to enable the use of data transformation and reductions as part of transformation parameters. E.g., a user may want to substract the minimum, or divide by the maximum etc.

Parameters

- **parameter** – The parameter to be evaluated. This may be a JSON string or list/dictionary-based description of a data transformation. Or any other valid data parameter. If the parameter describes as data reduction or transformation then the transformation will be evaluated and the result is returned, otherwise the parameter itself is returned.

- **data** – The input numpy array that should be transformed.

- **secondary_data** – Other data from previous data iterations a user may reference.

Returns The evaluated parameter result.

oms/shared/data_selection.is_transform_or_reduce(parameter)

Check if the given parameter defines a description of a data transformation or data reduction

Parameters **parameter** (JSON string, dict or list of dicts with transformation parameter.) – The parameter to be checked.
Returns Boolean

`oms_shared.data_selection.json_to_transform_reduce_description(json_string)`

Convert the json string to the transformation/reduction dict.

Parameters `json_string` – The json string to be converted.

Returns Python list or dict with the description

`oms_shared.data_selection.perform_reduction(data, reduction, secondary_data, min_dim=None, http_error=False, **kwargs)`

Helper function used reduce the data of a given numpy array.

Parameters

- `data` – The input numpy array that should be reduced
- `reduction` (String) – Data reduction to be applied to the input data. Reduction operations are defined as strings indicating the numpy function to be used for reduction. Valid reduction operations include e.g.: mins, max, mean, median, std, var etc.
- `axis` – The axis along which the reduction should be applied
- `secondary_data` – Other data from previous data iterations a user may reference.
- `http_error` – Define which type of error message the function should return. If false then None is returned in case of error. Otherwise a DJANGO HttpResponse is returned.
- `min_dim` – Minimum number of dimensions the input data must have in order for the reduction to be applied.
- `kwargs` – Additional optional keyword arguments.

Returns Reduced numpy data array or in case of error None or HttpResponse with a description of the error that occurred (see http_error option).


List of allowed numpy data reduction operations. Reduction operations are any single data operations that may change the shape of the data. NOTE: Some operations may have additional optional or required keyword arguments. HELP: For full documentation of the different functions see the numpy documentation.

Additional input parameters are often:

- `x1` [The data operand specifying the data the reduction should be performed on. \] The input data will be used by default if x1 is not specified. You may also specify ‘data’ to explicitly indicate that the input data should be assigned to x1. You may specify data0 to indicate that the output of another data operation should be used. Note, data0 here refers to the input to the full data operation pipeline. Data from other parts of the pipeline, are then indexed using 1-based indices. E.g., to access the output of the first data operation set x1=’data0’

- `axis` [Integer indicating the axis along which the data should be reduced. \] The default behavior, if axis is not specified, depends on the behavior of the corresponding numpy function. However, in most cases (if not all cases) the data operation will be applied to the full input data if no axis is specified.

- `min_dim` [Integer specifying the minimum number of data dimensions the input data \] must have in order for the reduction operation to be applied.

Here the list of allowed data reduction operations.

- ‘all’ : out = numpy.all(data, axis)
- ‘amax’ : out = numpy.amax(data, axis)
- ‘amin’ : out = numpy.amin(data, axis)
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

• 'alltrue' : out = numpy.alltrue(data, axis)
• 'angle' : out = numpy.angle(z, deg)
• 'any' : out = numpy.any(data, axis)
• 'append' : out = numpy.append(data, values, axis)
• 'argmax' : out = numpy.argmax(data, axis)
• 'argmin' : out = numpy.argmin(data, axis)
• 'argwhere' : out = numpy.argwhere(data)
• 'average' : out = numpy.average(data, axis)
• 'bincount' : out = numpy.bincount(x, weights=None, minlength=None)
• 'corrrho' : out = numpy.corrrho(data)
• 'count_nonzero' : out = numpy.count_nonzero(data)
• 'cumprod' : out = numpy.cumprod(data, axis)
• 'cumproduct': out = numpy.cumproduct(data, axis)
• 'cumsum' : out = numpy.cumsum(data, axis)
• 'diag' : out = numpy.diag(data, k=0)
• 'diag_indices: out = numpy.diag_indices(data, ndim=2)
• 'diagflat': out = numpy.diagflat(data, k=0)
• 'diagonal': out = numpy.diagonal(data, offset=0, axis1=0, axis2=1)
• 'diff': out = numpy.diff(a, n=1, axis=-1)
• 'max': out = numpy.max(data, axis)
• 'min': out = numpy.min(data, axis)
• 'median': out = numpy.median(data, axis)
• 'mean': out = numpy.mean(data, axis)
• 'percentile': out = numpy.percentile(data, q, axis)
• 'product': out = numpy.product(data, axis)
• 'prod': out = numpy.prod(data, axis)
• 'ptp': out = numpy.ptp(data, axis)
• 'squeeze': out = numpy.squeeze(data)
• 'std': out = numpy.std(data, axis)
• 'swapaxes: out = numpy.swapaxes(x1, axis1, axis2)
• 'var': out = numpy.var(data, axis)
• 'transpose': out = numpy.transpose(data)
• 'sum': out = numpy.sum(data, axis)

None-numpy data reduction operations:

• 'select_values' : out = data[ selection ]
Convert the given selection string to a python selection object, i.e., either a slice, list or integer index.

**Parameters**

- `selection_string` – A selection string of the type indexlist
- `list_to_index` – Should we turn the list into an index if the list contains only a single value. Default value is False, i.e., the list is not modified.

**Returns**

- An integer index if an index selection is specified
- A python list of indices if a list specified in the string
- A python slice object if a slice operation is specified by the string

Parse the indexlist selection string and return a python list of indices

**Parameters**

- `selection_string` – A selection string of the type indexlist
- `axis_size` – Size of the dimensions for which the selection is defined. Only needed in case that a range selection is given. This should be a list of sizes, in case that a multiaxis selection is given.

**Returns**

- A python list of point indices for the selection.
- None in case the list is empty or in case an error occurred.

Convert the given selection, which may be either an int, a list of ints, a slice object or a tuple of the mentioned types which is used to define a selection along multiple axes. :param selection: The selection to be converted to a string :type selection: int, list, slice, or a tuple of int, list, slice objects :return: The selection string

This an extended list of types indicated by the check_selection_string function. Indices <0 are assumed to be invalid selections.

Helper function used to apply a series of potentially multiple operations to a given numpy dataset. This function uses the transform_data_single(...) function to apply each indicated transformation to the data. This function uses the perform_reduction function to perform data reduction operations.

**Parameters**

- `data` – The input numpy array that should be transformed.
- `operations` – JSON string with list of dictionaries or a python list of dictionaries. Each dict specifies a single data transformation or data reduction. The operations are applied in order, i.e., operations[0] is applied first, then operations[1] and so on. The dicts must be structured according to one of the following specifications:
  - `{‘transformation’:<op>}`: Single transformation applied to all data at once.
  - `{‘transformation’:<op>, ‘axes’:[..]}`: Apply a single transformation to data chunks defined by the axes parameter. The data is split into chunks along the dimensions defined.
by the axes parameter. E.g., if we have a 3D MSI dataset and we want to op ion images independently, then we need to set axes=[2]. Accordingly, if we want to op spectra individually, then we need to split the two image dimensions into chunks by setting axes=[0,1].

- `{reduction:<reduction>, 'axis':int}`: Define the reduction operations to be applied and the axis along which the data should be reduced. If reduction along all axis should be done then set axis ot None (in python) or null in JSON.

- **secondary_data** – Other data from previous data iterations a user may reference.
- **http_error** – Define which type of error message the function should return. If false then None is returned in case of error. Otherwise a DJANGO HttpResponse is returned.

**Returns** Reduced numpy data array or HttpResonse with a description of the error that occurred.

```python
omsia.shared.data_selection.transform_data_single(data, transformation='minusMinDivideMax', axes=None, secondary_data=None, http_error=False, transform_kwargs=None)
```

Helper function used to transform data of a numpy array. The function potentially splits the array into independent chunks that are normalized separately (depending on how the axes parameter is defined). The actual data transformations are implemented by transform_datachunk(...).

**Parameters**

- **data** – The input numpy array that should be transformed.
- **transformation** – Data transformation option to be used. Available options are: ‘minusMinDivideMax’, ‘...’
- **axes** – List of data axis that should be split into chunks that are treated independently during the transformation. By default transformation is applied based on the full dataset (axes=None). E.g, if transformation should be performed on a per image basis, then we need to split the m/z dimension into individual chunks and set axes=[2]. If we want to transform spectra individually, then we need to split the two image dimensions into chunks by setting axes=[0,1].
- **secondary_data** – Other data from previous data iterations a user may reference.
- **http_error** – Define which type of error message the function should return. If false then None is returned in case of error. Otherwise a DJANGO HttpResponse is returned.
- **transform_kwargs** – Dictionary of additional keyword arguments to be passed to the transform_datachunk(...) function.

**Returns** Reduced numpy data array or HttpResonse with a description of the error that occurred.

```python
omsia.shared.data_selection.transform_datachunk(data, transformation='minusMinDivideMax', secondary_data=None, **kwargs)
```

Helper function used to transform a given data chunk. In contrast to transform_data, this function applies the transformation directly to the data provided, without consideration of axis information. This function is used by transform_data(...) to implement the actual normalization for independent data chunks that need to be normalized.

**Required keyword arguments:**

**Parameters**

- **data** – The input numpy array that should be transformed.
• **transformation** – Data transformation option to be used. For available options see the transformation_type dictionary.

• **secondary_data** – Other data from previous data iterations a user may reference.

Additional transformation-dependent keyword arguments:

**Parameters** `kwargs` – Additional keyword arguments that are specific for different data transformation. Below a list of additional keyword arguments used for different transformation options

• **transformation**: ‘threshold’

  **‘threshold’** [The threshold parameter to be used for] the threshold operation. If threshold is not specified, then the 5th %tile will be used as threshold value instead, i.e., the bottom 5% of the data are set to 0.

**Returns** This function returns the normalized data array. If an unsupported transformation option is given, then the function simply return the unmodified input array.

```python
omsi.shared.data_selection.transform_reduce_description_to_json(*args)
```

Convert the dictionary describing the transformation/reduction operations to a JSON string.

**Parameters** `args` – The list or dictionaries with the description of the transformation and reduction operations.

**Returns** JSON string

```python
omsi.shared.data_selection.transformation_allowed_numpy_dual_data=['add', 'arctan2', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor', 'corrcoef', 'cov', 'divide', 'equal', ...
```

List of allowed dual data transformations. Dual data transformation, are operation that operate on a two data input datasets but which do not change the shape of the data. Below a list of available numpy function options. NOTE: Some operations may have additional optional or required keyword arguments. HELP: For full documentation of the different functions see the numpy documentation.

• `'add'` : out = x1 + x2 = numpy.add(x1,x2)

• `'arctan2'` : out = numpy.arctan2(x1,x2)

• `'bitwise_and'` : out = x1 && x2 = numpy.bitwise_and(x1,x2)

• `'bitwise_not'` : out = numpy.bitwise_not(x1,x2)

• `'bitwise_or'` : out = x1 || x2 = numpy.bitwise_or(x1,x2)

• `'bitwise_xor'` : out = numpy.bitwise_xor(x1,x2)

• `'corrcoef'` : out = numpy.corrcoef(x1,x2)

• `'cov'` : out = numpy.cov(x1, x2, rowvar=1, bias=0, ddof=None)

• `'divide'` : out = x1 / x2 = numpy.divide(x1,x2)

• `'equal'` : out = x1 == x2 = numpy.equal(x1,x2)

• `'fmax'` : out = numpy.fmax(x1,x2)

• `'fmin'` : out = numpy.fmin(x1,x2)

• `'fmod'` : out = numpy.fmod(x1,x2)

• `'greater'` : out = x1 > x2 = numpy.greater(x1,x2)

• `'greater_equal'` : out = x1 >= x2 = numpy.greater_equal(x1,x2)

• `'left_shift'` : out = numpy.left_shift(x1,x2)

• `'less'` : out = x1 < x2 = numpy.less(x1,x2)

• `'less_equal'` : out = x1 <= x2 = numpy.less_equal(x1,x2)

10.1. Subpackages 193
• 'logical_and': out = numpy.logical_and(x1, x2)
• 'logical_not': See transformation_allowed_numpy_single_data instead.
• 'logical_or': out = numpy.logical_or(x1, x2)
• 'logical_xor': out = numpy.logical_xor(x1, x2)
• 'mod': out = numpy.mod(x1, x2)
• 'multiply': out = x1 * x2 = numpy.multiply(x1, x2)
• 'not_equal': out = x1 != x2 = numpy.not_equal(x1, x2)
• 'power': out = numpy.power(x1, x2)
• 'subtract': out = x1 - x2 = numpy.subtract(x1, x2)
• 'right_shift': out = np.right_shift(x1, x2)

oms.shared.data_selection.transformation_allowed_numpy_single_data = ['abs', 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctanh', 'argsort', 'around', 'ceil', 'clip', 'cos', ...
• 'log': out[x1>0] = log(x1[x1>0]) \] out[x1<0] = log(x1[x1<0]*-1)*-1 out[x1==0] = 0
\*\*log2 \*\* `out[x1>0] = log2(x1[x1>0])` \*\*log10` out[x1>0] = log10(x1[x1>0])` out[x1<0] = log10(x1[x1<0]*-1)*-1 out[x1==0] = 0
\*\*logical_not` out = numpy.logical_not(x1)
\*\*negative` out = np.negative(x1)
\*\*round` out = numpy.round(x1, decimals)
\*\*sqrt` [out[x1>0] = sqrt(x1[x1>0]) \] out[x1<0] = sqrt(x1[x1<0]*-1)*-1 out[x1==0] = 0
\*\*sign` out = numpy.sign(x1)
\*\*sin` out = numpy.sin(x1)
\*\*sinc` out = numpy.sinc(x1)
\*\*sinh` out = numpy.sinh(x1)
\*\*sort` out = numpy.sort(x1, axis=-1, kind='quicksort', order=None)
\*\*swapaxes` out = numpy.swapaxes(x1, axis1, axis2)
\*\*tan` out = numpy.tan(x1)
\*\*tanh` out = numpy.tanh(x1)

```
omsi.shared.data_selection.transformation_type = {'singleDataTransform': 'singleDataTransform', 'scale': 'scale', 'divideMax': 'divideMax', 'astype': 'astype', 'threshold': 'threshold', 'minusMinDivideMax': 'minusMinDivideMax', 'dualDataTransform': 'dualDataTransform', 'arithmetic': 'arithmetic'}
```

Dictionary of available data transformation options. Available options are:

- `arithmetic`: Same as ‘dualDataTransform’. See ‘dualDataTransform’ below for details.
- `divideMax`: Divide the data by the current maximum value.
- `minusMinDivideMax` [Subtract the minimum value from the data and \] then divide the data by maximum of the data (with the minimum already substracted.)
- `dualDataTransform` [Apply arbitrary arithmetic operation to the data. Additional parameter \] required for this option are:
  - `operation`: String defining the arithmetic operations to be applied. Supported operations are: ‘add’, ‘divide’, ‘greater’, ‘greater_equal’, ‘multiply’, ‘subtract’
  - `x1` [The first data operand of the arithmetic operation. \] The input data will be used by default if x1 is not specified. You may also specify ‘data’ to explicitly indicate that the input data should be assigned to x1. You may specify data0 to indicate that the output of another data operation should be used. Note, data0 here refers to the input to the full data operation pipeline. Data from other parts of the pipeline, are then indexed using 1-based indices. E.g., to access the output of the first data operation set x1='data0'
  - `x2` [The second data operand of the arithmetic operation. \] The input data will be used by default if x2 is not specified. You may also specify ‘data’ to explicitly indicate that the input data should be assigned to x2. You may specify data0 to indicate that the output of another data operation should be used. Note, data0 here refers to the input to the full data operation pipeline. Data from other parts of the pipeline, are then indexed using 1-based indices. E.g., to access the output of the first data operation set x2='data0'
  - ... any additional parameters needed for the numpy function.
- `singleDataTransform` [Apply scaling transformation to the data. Additional parameters \] required for this options are. NOTE: operation==’log or operation==’sqrt’: If the minimum value is 0 then the transformation is applied to positive values only and 0 values remain as is. If the minimum value is larger then 0, then the log-scale is applied as is, i.e., np.log(data). If the minimum data value is
negative, then the log scale is applied independently to the positive values and the negative values, ie.,

- ‘operation’ : String defining the scaling operations to be applied. See the transforma-
tion_allowed_numpy_single_data list for a complete list of allowed scaling operations. Some
of the more commonly used scaling operations include: ‘abs’, ‘log’, ‘sqrt’, ‘around’ etc.

- ‘x1’ [The first data operand for the scaling.] The input data will be used by default if x1 is
not specified. You may also specify ‘data’ to explicitly indicate that the input data should be
assigned to x1.

Additional optional keyword arguments depending on the used operation:

- ‘decimals’ [Number of decimal places to round to when using numpy.around or numpy.round
\] (default: 0). If decimals is negative, it specifies the number of positions to the left of the
decimal point.

- ‘a_min’, ‘a_max’ : Lower and upper bound when using numpy.clip.


- ...

•‘scale’ : Same as ‘singleDataTransform’. See ‘singleDataTransform’ for details.

•‘threshold’ [Threshold the data. Set all values that are smaller than threshold \] to 0. Additional parame-
ters required for this option are:

  - ‘threshold’. If threshold is missing, then the threshold will be
    set ot the 5’ile so that the bottom 5% of the data will be set to 0.

•‘astype’ : Change the type of the data. Additional required parameters are:

  - ‘dtype’ : The numpy data type to be used. Default dtype=’float’.

omsi_web_helper Module

Module with helper functions for interactions with the OpenMSI web infrastructure, e.g. update job status, explicitly
add a file to the OpenMSI database, update file permissions so that Apache can access it etc.

class omsi.shared.omsi_web_helper.UserInput
    Bases: object

    Collection of helper functions used to collect user input

    static userinput_with_timeout (timeout, default=’’)
    Read user input. Return default value given after timeout. This function decides which platform-dependent
    version should be used to retrieve the user input.

        Parameters

        • timeout – Number of seconds till timeout

        • default (String) – Default string to be returned after timeout

        Returns  String

    static userinput_with_timeout_default (timeout, default=’’)
    Read user input. Return default value given after timeout.

        Parameters
• **timeout** – Number of seconds till timeout

• **default** *(String)* – Default string to be returned after timeout

Returns String

```python
static userinput_with_timeout_windows(timeout, default='')
```

Read user input. **Return default value given after timeout.** This function is used when running on windows-based systems.

Parameters

• **timeout** – Number of seconds till timeout

• **default** *(String)* – Default string to be returned after timeout

Returns String

```python
class omsi.shared.omsi_web_helper.WebHelper
    Bases: object

    Class providing a collection of functions for web-related file conversion tasks, e.g. i) adding files to the web database, ii) notifying users via email, iii) setting file permissions for web-access.

    allowed_nersc_locations = ['/project/projectdirs/openmsi/omsi_data_private', '/global/project/projectdirs/openmsi/omsi_data_private', '/data/openmsi/omsi_data']

    default_db_server_url = 'https://openmsi.nersc.gov/

    static register_file_with_db(filepath, db_server, file_user_name, jobid=None, check_add_nersc=True)

    Function used to register a given file with the database

    Parameters

    • **filepath** – Path of the file to be added to the database

    • **db_server** – The database server url

    • **file_user_name** – The user to be used, or None if the user should be determined based on the file URL.

    • **jobid** – Optional input parameter defining the jobid to be updated. If the jobid is given then the job will be updated with the database instead of adding the file explicitly. I.e., instead of register_file_with_db the update_job_status call is executed.

    Returns Boolean indicating whether the operation was successful

    static send_email(subject, body, sender='convert@openmsi.nersc.gov', email_type='success', email_success_recipients=None, email_error_recipients=None)

    Send email notification to users.

    Parameters

    • **subject** – Subject line of the email

    • **body** – Body text of the email.

    • **sender** – The originating email address

    • **email_type** – One of ‘success’, ‘error’, ‘warning’. Error messages are sent to ConvertSettings.email_error_recipients, success messages to ConvertSettings.email_success_recipients and warning messages are sent to both lists.

    • **email_success_recipients** – List of user that should receive an email if the status is success or warning.
• **email_error_recipients** – List of users that should receive an email if the status is error or warning.

    static set_apache_acl (filepath)
    Helper function used to set acl permissions for apache to make the given file accesible to Apache at NERSC. This necessary to make the file readable for adding it to the database.

    super_users = ['bpb', 'oruebel']

    static update_job_status (filepath, db_server, jobid, status='complete')
    Function used to update the status of the job on the server

    Parameters
    • **filepath** – Path of the file to be added to the database (only needed update file permissions)
    • **db_server** – The database server url
    • **jobid** – The id of the current job.
    • **status** – One of ‘running’, ‘complete’ or ‘error’

**spectrum_layout Module**

This module provides capabilities for computing different layouts for spectra

    omsi.shared.spectrum_layout.compute_hilbert_spectrum (original_coords, original_intensities, left=0, right=0)

    Given a 1D spectrum, interpolate the spectrum onto the closest 2D hilbert curve.

    Parameters
    • **original_coords** (*1D numpy array in increasing order.*) – The original coordinate values (m/z). Values must be increasing.
    • **original_intensities** (*1D numpy array of same length as original_coords*) – The original intensity values. Same length as original_coords.
    • **left** – Optional. Value to be used for padding data at the lower bound during interpolation
    • **right** – Optional. Value to be used for padding data at the upper bound during interpolation

    Type float
    Type float

    Returns 2D numpy array with the coordinate (m/z) values for the hilbert spectrum and separate 2D numpy array for the interpolated intensity values.

    Raises ValueError If original_coords and original_intensities have different length.

    omsi.shared.spectrum_layout.hilbert_curve (order=2)
    Compute a 2D hilbert curve.

    Parameters **order** (*Integer that defines a power of 2 (>=2]*) – The order of the hilber curve. This is the length of the sides of the square, i.e., the number of points in x and y.

    Returns Returns two numpy arrays of integers x,y, indicating the locations of the vertices of the hilbert curve.
omsis.shared.spectrum_layout.plot_2d_spectrum_as_image(hilbert_intensities, show_plot=False, show_axis=False)

Plot image with pixels colored according to hilbert_intensities.

Parameters

• hilbert_intensities (2D numpy array.) – 2D numpy array with the intensity values for the spectrum.
• show_plot (Boolean) – Show the generated plot in a window.
• show_axis (Boolean) – Show x,y axis for the plot. Default is False.

Returns matplotlib image plot or None in case that the plotting failed.

omsis.shared.spectrum_layout.reinterpolate_spectrum(coords, original_coords, original_intensities, left=0, right=0)

Given a 1D spectrum, interpolate the spectrum onto a new axis.

Parameters

• coords – The coordinate values (m/z) for which intensities should be computed.
• original_coords – The original coordinate values (m/z). Values must be increasing.
• original_intensities – The original intensity values. Same length as original_coords.
• left – Optional. Value to be used if coords < original_coords
• right – Optional. Value to be used if coords > original_coords

Returns y : {float, ndarray} The interpolated values, same shape as coords.

Raises ValueError If original_coords and original_intensities have different length.

log Module

Module providing functionality for logging based on the python logging module. The module is intended to ease the use of logging while a developer can still access the standard python logging mechanism if needed.

class omsis.shared.log.log_helper
Bases: object

BASTet helper module to ease the use of logging

Class Variables:

Variables log_levels – Dictionary describing the different available logging levels.

classmethod critical(module_name, message, root=0, comm=None, *args, **kwargs)
Create a critical log entry. This function is typically called as:
log_helper.critical(module_name=__name__, message="your message")

Parameters

• module_name – __name__ of the calling module or None in case the ROOT logger should be used.
• message – The message to be added to the log
• root – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.
• **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

• **args** – Additional positional arguments for the python logger.debug function. See the python docs.

• **kwargs** – Additional keyword arguments for the python logger.debug function. See the python docs.

```python
classmethod debug(module_name, message, root=0, comm=None, *args, **kwargs)
```
Create a debug log entry. This function is typically called as:

```python
log_helper.debug(module_name=__name__, message="your message")
```

**Parameters**

• **module_name** – __name__ of the calling module or None in case the ROOT logger should be used.

• **message** – The message to be added to the log

• **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

• **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

• **args** – Additional positional arguments for the python logger.debug function. See the python docs.

• **kwargs** – Additional keyword arguments for the python logger.debug function. See the python docs.

```python
classmethod error(module_name, message, root=0, comm=None, *args, **kwargs)
```
Create an error log entry. This function is typically called as:

```python
log_helper.error(module_name=__name__, message="your message")
```

**Parameters**

• **module_name** – __name__ of the calling module or None in case the ROOT logger should be used.

• **message** – The message to be added to the log

• **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

• **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

• **args** – Additional positional arguments for the python logger.debug function. See the python docs.

• **kwargs** – Additional keyword arguments for the python logger.debug function. See the python docs.

```python
classmethod exception(module_name, message, root=0, comm=None, *args, **kwargs)
```
Create a exception log entry. This function is typically called as:

```python
log_helper.exception(module_name=__name__, message="your message")
```

**Parameters**

• **module_name** – __name__ of the calling module or None in case the ROOT logger should be used.
• **message** – The message to be added to the log

• **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

• **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

• **args** – Additional positional arguments for the python logger.debug function. See the python docs.

• **kwargs** – Additional keyword arguments for the python logger.debug function. See the python docs.

```python
classmethod get_default_format()
    Get default formatting string.

classmethod get_logger(module_name)
    Get the logger for a particular module. The module_name should always be set to the __name__ variable of the calling module.

    Parameters
    module_name -- __name__ of the calling module or None in case the ROOT logger should be used.

    Returns
    Python logging.Logger retrieved via logging.getLogger.

global_log_level = 20

classmethod info(module_name, message, root=0, comm=None, *args, **kwargs)
    Create a info log entry. This function is typically called as:

    log_helper.info(module_name=__name__, message="your message")

    Parameters

    • **module_name** -- __name__ of the calling module or None in case the ROOT logger should be used.

    • **message** – The message to be added to the log

    • **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

    • **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

    • **args** – Additional positional arguments for the python logger.debug function. See the python docs.

    • **kwargs** – Additional keyword arguments for the python logger.debug function. See the python docs.

initialized = False

classmethod log(module_name, message, root=0, comm=None, level=None, *args, **kwargs)
    Convenience function used to select the log message level using an input parameter rather than by selecting the appropriate function.

    Parameters

    • **module_name** -- __name__ of the calling module or None in case the ROOT logger should be used.

    • **message** – The message to be added to the log
• **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

• **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

• **level** – To which logging level should we send the message

• **args** – Additional positional arguments for the python logger.debug function. See the python docs.

• **kwargs** – Additional keyword arguments for the python logger.debug function. See the python docs.

\[
\]

```python
classmethod log_var(module_name, root=0, comm=None, level=None, **kwargs)

Log one or more variable values
```

**Parameters**

- **module_name** — __name__ of the calling module or None in case the ROOT logger should be used.

- **message** – The message to be added to the log

- **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

- **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

- **kwargs** – Variables+values to be logged

```python
classmethod set_log_level(level)

Set the logging level for all BASTet loggers
```

**Parameters**

- **level** – The logging levels to be used, one of the values specified in log_helper.log_levels.

```python
classmethod setup_logging(level=None)

Call this function at the beginning of your code to initiate logging.
```

**Parameters**

- **level** – The default log level to be used. One of log_helper.log_level.

```python
classmethod warning(module_name, message, root=0, comm=None, *args, **kwargs)

Create a warning log entry. This function is typically called as:

log_helper.warning(module_name=__name__, message="your message")
```

**Parameters**

- **module_name** — __name__ of the calling module or None in case the ROOT logger should be used.

- **message** – The message to be added to the log

- **root** – The root process to be used for output when running in parallel. If None, then all calling ranks will perform logging. Default is 0.

- **comm** – The MPI communicator to be used to determine the MPI rank. None by default, in which case mpi.comm_world is used.

- **args** – Additional positional arguments for the python logger.debug function. See the python docs.
• `kwargs` – Additional keyword arguments for the python logger.debug function. See the python docs.

**mpi_helper Module**

Module used to ease the use of MPI and distributed parallel implementations using MPI

```python
oms.shared.mpi_helper.barrier (comm=None)
```

MPI barrier operation or no-op when running without MPI

**Parameters**

- `comm` – MPI communicator. If None, then MPI.COMM_WORLD will be used.

```python
oms.shared.mpi_helper.broadcast (data, comm=None, root=0)
```

MPI broadcast operation to broadcast data from one rank to all other ranks

**Parameters**

- `data` – The data to be gathered
- `comm` – MPI communicator. If None, then MPI.COMM_WORLD will be used.
- `root` – The rank where the data is send from

**Returns**

The data object

```python
oms.shared.mpi_helper.gather (data, comm=None, root=0)
```

MPI gather operation or return a list with just [data,:) if MPI is not available

**Parameters**

- `data` – The data to be gathered
- `comm` – MPI communicator. If None, then MPI.COMM_WORLD will be used.
- `root` – The rank where the data should be collected to. Default value is 0

**Returns**

List of data objects from all the ranks

```python
oms.shared.mpi_helper.get_comm_world ()
```

Get MPI.COMM_WORLD :return: mpi communicator or None if MPI is not available

```python
oms.shared.mpi_helper.get_rank (comm=None)
```

Get the current process rank :param comm: MPI communicator. If None, then MPI.COMM_WORLD will be used. :return: The integer index of the rank

```python
oms.shared.mpi_helper.get_size (comm=None)
```

Get the size of the current communication domain: :param comm: MPI communicator. If None, then MPI.COMM_WORLD will be used. :return: The integer index of the rank

```python
oms.shared.mpi_helper.imports_mpi (python_object)
```

Check whether the given class import mpi

The implementation inspects the source code of the analysis to see if MPI is imported by the code.

```python
oms.shared.mpi_helper.is_mpi_available ()
```

Check if MPI is available. Same as MPI_AVAILABLE :return: bool indicating whether MPI is available

```python
oms.shared.mpi_helper.mpi_type_from_dtype (dtype)
```

Get the corresponding MPI type for the given basic numpy dtype

**Parameters**

- `dtype` – Basic numpy dtype to be mapped to the MPI type

**Returns**

The MPI type or None if not found
class omsi.shared.mpi_helper.parallel_over_axes (task_function, task_function_params, main_data, split_axes, main_data_param_name, schedule='STATIC_1D', root=0, comm=None)

Bases: object

Helper class used to parallelize the execution of a function using MPI by splitting the input data into sub-blocks along a given set of axes.

Variables

- **task_function** – The function we should run.
- **task_function_params** – Dict with the input parameters for the function. May be None or {} if no parameters are needed.
- **main_data** – Dataset over which we should parallelize
- **split_axes** – List of integer axis indices over which we should parallelize
- **main_data_param_name** – The name of data input parameter of the task function
- **root** – The master MPI rank (Default=0)
- **schedule** – The task scheduling schema to be used (see parallel_over_axes.SCHEDULES
- **collect_output** – Should we collect all the output from the ranks on the master rank?
- **schedule** – The parallelization schedule to be used. See also parallel_over_axes.schedule
- **result** – The result form the task_function. If self.__data_collected is set and we are the root then this will a list with the the output of all tasks
- **blocks** – List with tuples describing the selected subset of data processed by the given block task. If self.__data_collected is set and we are the root rank then this is a list of all the blocks processed by each rank.
- **block_times** – List of times in seconds used to process the data block with the given index. NOTE: The block times include also any required communications and other operations to initialize and complete the task, and not just the execution of the task function itself.
- **run_time** – Float time in seconds for executing the run function.
- **comm** – The MPI communicator used for the parallelization. Default value is MPI.COMM_WORLD

Parameters

- **task_function** – The function we should run.
- **task_function_params** – Dict with the input parameters for the function. May be None or {} if no parameters are needed.
- **main_data** – Dataset over which we should parallelize
- **split_axes** – List of integer axis indices over which we should parallelize
- **main_data_param_name** – The name of data input parameter of the task function
- **root** – The master MPI rank (Default=0)
- **schedule** – The task scheduling schema to be used (see parallel_over_axes.SCHEDULES
- **comm** – The MPI communicator used for the parallelization. Default value is None, in which case MPI.COMM_WORLD is used
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

MPI_MESSAGE_TAGS = {'BLOCK_MSG': 12, 'COLLECT_MSG': 13, 'RANK_MSG': 11}
SCHEDULES = {'DYNAMIC': 'DYNAMIC', 'STATIC_ID': 'STATIC_ID', 'STATIC': 'STATIC'}

collect_data(force_collect=False)
Collect the results from the parallel execution to the self.root rank.

NOTE: On the root the self.result, self.blocks, and self.block_times variables are updated with the collected data as well and self.__data_collected will be set

NOTE: If the data has already been collected previously (ie., collect_data has been called before), then the collection will not be performed again, unless force_collect is set.

Parameters force_collect – Set this parameter to force that data collection is performed again. By default the collect_data is performed only once for each time the run(..) function is called and the results are reused to ensure consistent data structures. We can force that collect will be reexecuted anyways by setting force_collect.

Returns On worker ranks (i.e., MPI_RANK!=self.root) this is simply the self.result and self.blocks containing the result created by run function. On the root rank (i.e., MPI_RANK!=self.root) this is a tuple of two lists containing the combined data of all self.result and self.blocks from all ranks respectively.

run()
Call this function to run the function in parallel.

Returns
Tuple with the following elements:
1. List with the results from the local execution of the task_function. Each entry is the result from one return of the task_function. In the case of static execution, this is always a list of length 1.

2. List of block_indexes. Each block_index is a tuple with the selection used to divide the data into sub-blocks. In the case of static decomposition we have a range slice object along the axes used for decomposition whereas in the case of dynamic scheduling we usually have single integer point selections for each task.

oms_shared.mpi_helper.test_mpi_available()
This function import MPI in a separate process to safely check if MPI is available. This precaution is necessary as on Cray systems importing MPI can lead to a crash on, e.g., login nodes where the use of MPI is not permitted. By executing the import in a separate process we avoid crashing the main process and we can safely check whether the process aborted or not.

Returns False if the import failed, otherwise return True

10.1.5 workflow Package

10.1. Subpackages 205

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omsi.workflow</td>
<td>Package with modules for specification and execution of analysis tasks and workflows.</td>
</tr>
<tr>
<td>omsi.workflow.common</td>
<td>Module defining basic data structures used by workflows.</td>
</tr>
<tr>
<td>omsi.workflow.driver</td>
<td>Package with drivers for analyses and workflows.</td>
</tr>
<tr>
<td>omsi.workflow.driver.base</td>
<td>Module with base classes for workflow drivers.</td>
</tr>
<tr>
<td>omsi.workflow.driver.cl_analysis_driver</td>
<td>Module used to help with driving the execution of omsi-based analyses.</td>
</tr>
<tr>
<td>omsi.workflow.driver.cl_workflow_driver</td>
<td>Module used to help with driving the execution of analysis workflows.</td>
</tr>
<tr>
<td>omsi.workflow.executor</td>
<td>Package with executors of analysis workflows.</td>
</tr>
</tbody>
</table>
workflow Package

Package with modules for specification and execution of analysis tasks and complex analysis workflows.

common Module

Module defining basic data structures used by workflows.

```python
class omsi.workflow.common.RawDescriptionDefaultHelpArgParseFormatter:
    Bases: argparse.ArgumentDefaultsHelpFormatter, argparse.RawDescriptionHelpFormatter
    Simple derived formatter class for use with argpase used by the cl_analysis_driver class. This formatter combines the default argparse.ArgumentDefaultsHelpFormatter and argparse.RawDescriptionHelpFormatter for formatting arguments and help descriptions.

class omsi.workflow.common.analysis_task_list:
    Bases: list
    Define a python list of analyses to be executed as a workflow. The list allows only for storage of analysis_base objects and ensures uniqueness of elements in the list.
    Initialize the set of analysis tasks to be performed as part of the workflow
    Parameters analysis_objects – Set or list of unique analysis objects to be added. Duplicates will be removed.
    add(analysis_object)  # Same as append
    add_all()  # Add all known analyses to the workflow list
    Returns The updated analysis_task_list with all analyses added
    add_analysis_dependencies()  # Add the dependencies of all analyses to the workflow list in case they are missing.
        This function is recursive, step-by-step adding all dependencies of the workflow to the list of tasks to be executed, until no more dependencies are found.
        Usually this function is called by the workflow executor itself before running the analysis and should not need to be called by the user.
        Returns Integer indicating the number of dependencies added to the list of tasks
    @classmethod all()  # Get an analysis_task_list of all current analysis_base objects
        Returns New analysis_task_list of all current analysis objects
    analysis_identifiers_unique()  # Check whether all identifiers of the analyses in the this list are unique. :return: bool
```
append(analysis_object)
Add a given analysis to the set of object to be executed by the workflow

This is the same as set.add() but we ensure that only analysis_base objects are added.

Parameters analysis_object (omsi.analysis.base.analysis_base) – Analysis object to be added to the execution. All dependencies of the analysis will also be executed as part of the execution.

Raises ValueError is raised if the given analysis_object is invalid

clear()
Remove all elements from the list

enable_memory_profiling(enable=True)
Enable or disable line-by-line profiling of memory usage of execute_analysis.

Parameters enable_memory (bool) – Enable (True) or disable (False) line-by-line profiling of memory usage

Raises ImportError is raised if a required package for profiling is not available.

enable_time_and_usage_profiling(enable=True)
Enable or disable profiling of time and usage of code parts of execute_analysis for all analyses.

Parameters enable (bool) – Enable (True) or disable (False) profiling

Raises ImportError is raised if a required package for profiling is not available.

classmethod from_script_files(script_files)
Same as from_script, but the script is read from the given files.

Parameters script_files – List of strings with the paths to the script files. If only a single script is used, then a single string may be used as well.

Returns An instance of workflow_base with the specification of the workflow to be executed

classmethod from_scripts(scripts)
Evaluate the workflow script to extract all analyses to be run.

NOTE: This function executes using eval(..), i.e., there are NO safeguards against malicious codes.

Parameters scripts – The script with the setup of the workflow. This should only include the definition of analyses and their inputs.

Returns An instance of workflow_base with the specification of the workflow to be executed

get_additional_analysis_dependencies()
Compute a list of all dependencies of the current list of analyses (excluding analyses that are already in the list of tasks).

Returns analysis_task_list of all analysis dependencies

get_all_analysis_data()
Get a list of all output data objects for all analysis

Returns List of omsi.analysis.analysis_data.analysis_data objects, one for each analysis

get_all_analysis_identifiers()
Get a list of all analysis identifiers

Returns List of strings with the analysis identifier of each analysis

get_all_dependency_data()
Get the complete list of all direct and indirect dependencies of all analysis tasks.
NOTE: These are only the direct dependencies as specified by the analysis itself. Use get_all_dependency_data_recursive(..) to also get the indirect dependencies of the analysis due to dependencies of the dependencies themselves.

**Returns** List of parameter_data objects that define dependencies.

**get_all_parameter_data** *(exclude_dependencies=False)*

Get the complete list of all parameters

**Parameters** exclude_dependencies – Boolean indicating whether we should exclude parameters that define dependencies from the list

**Returns** List of omsi.analysis.analysis_data.parameter_data objects with the description of the parameters

**get_all_run_info** ()

Get a list of dict with the complete info about the last run of each of the analysis

**Returns** List of run_info_dict objects, one for each analysis

**insert** *(index, analysis_object)*

Insert a given analysis object at the given location

**Parameters**

- **index** – Location where the object should be inserted
- **analysis_object** – The analysis object to be inserted

**make_analysis_identifiers_unique** ()

Update analysis identifiers to be unique.

Side effects: This function updates the analysis tasks stored in the set

**Returns** self, i.e., the modified object with identifiers updated

**set_undefined_analysis_identifiers** ()

Check that all analysis descriptors are set to a value different than “undefined” and set the descriptor based on their index in the list if necessary.

**update** *(analysis_objects)*

Return the set with elements added from the given set of analysis_objects.

This is the same as set.update() but we ensure that only analysis_base objects are added.

**Parameters** analysis_objects – List or set of analysis_base objects to be added to workflow set

**Raise** ValueError is raised in case that objects that are not instances of analysis_base are to be added.

**Returns** self with elements added to self.

**Subpackages**

**driver Package**

**driver Package** Package with drivers for analyses and workflows. Drivers are intended to control the set up —i.e., creation and initialization—of workflows. The execution of workflows itself is typically performed by a workflow executor.
**base Module** Module with base classes for workflow drivers.

Workflow drivers are responsible for the creation and initialization of workflows. The execution of workflows is then controlled by the workflow executor.

```python
class omsi.workflow.driver.base.analysis_driver_base(analysis_class)
    Bases: omsi.workflow.driver.base.driver_base

    Base class defining the minimal interface for drivers of a single analysis based on the type/class of the analysis.
    This is a class-based execution, i.e, the user defines only the type of analysis and inputs but does not actually create the analysis.
    Derived classes must implement the main(...) function where the analysis is created and executed.

    Variables `analysis_class` – The analysis class for which we want to execute the analysis. The analysis class must derive from omsi.analysis.analysis_base. May be None in case that we use other means to set the analysis_class, e.g., via the command-line.

Initialize the analysis driver

    Variables `analysis_class` – The analysis class for which we want to execute the analysis. The analysis class must derive from omsi.analysis.analysis_base. May be None in case that we use other means to set the analysis_class, e.g., via the command-line.

    main()
    The main function for running the analysis.

class omsi.workflow.driver.base.driver_base
    Bases: object

    Primitive base class for driving the execution of an object

    execute()
    Same as main

    main()
    The main function for running the analysis.

class omsi.workflow.driver.base.workflow_driver_base(workflow_executor)
    Bases: object

    Base class defining the minimal interface for drivers of complex analysis workflows.
    Workflows may be specified via scripts or given via a set of analysis objects.
    Derived classes must implement the main(...) function where the analysis is created and executed.

    Variables `workflow_executor` – The executor of the workflow.

    Initialize the workflow driver

    Parameters `workflow_executor` – The executor of the workflow we want to drive.

    main()
    The main function for running the analysis.
```

**cl_analysis_driver Module** Module used to help with driving the execution of omsi-based analyses.

```python
class omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver(analysis_class,
    add_analysis_class_arg=False,
    add_output_arg=True,
    add_log_level_arg=True)

    Bases: omsi.workflow.driver.base.analysis_driver_base
```

10.1. Subpackages 209
Command-line analysis driver.

Variables

- **analysis_class_arg_name** – The name for the argument where the positional argument for defining the analysis class should be stored.

- **output_save_arg_name** – Name of the optional keyword argument for specifying the name and target for the analysis

- **analysis_class_arg_name** – Name of the optional first positional argument to be used to define the analysis class to be used.

- **log_level_arg_name** – Name of the keyword cl argument to define the level of logging ot be used.

- **analysis_class** – The class (subclass of analysis_base) defining the analysis to be executed

- **analysis_object** – Instance of the analysis object to be executed

- **add_analysis_class_arg** – Boolean indicating whether an optional positional command line argument should be used to determine the analysis class (or whether the analysis class will be set explicitly)

- **add_output_arg** – Boolean indicating whether an optional keyword argument should be added to define the output target for the analysis.

- **add_log_level_arg** – Boolean indicating whether the –loglevel argument should be added to the command line

- **parser** – The argparse.ArgumentParser instance used for defining command-line arguments

- **required_argument_group** – argparse.ArgumentParser argument group used to define required command line arguments

- **custom_argument_groups** – Dict of custom argparse.ArgumentParser argument groups specified by the analysis

- **output_target** – Specification of the output target where the analysis result should be stored

- **analysis_arguments** – Dictionary defining the input arguments to be used for the analysis

- **mpi_root** – Integer indicating the root rank when running using MPI

- **mpi_comm** – Integer indicating the MPI communicator to be used when running in parallel using MPI

Parameters

- **analysis_class** (omsi.analysis.base.analysis_base) – The analysis class for which we want to execute the analysis. The analysis class must derive from omsi.analysis.analysis_base. May be None in case that we use the command-line to define the analysis class via the optional positional argument for the command class (i.e., set add_analysis_class_arg to True).

- **add_analysis_class_arg** – Boolean indicating whether we will use the positional command-line argument to determine the analysis class name

- **add_output_arg** – Boolean indicating whether we should add the optional keyword argument for defining the output target for the analysis.
• **add_log_level_arg** – Boolean indicating whether we should add the option keyword argument to specify the logging level via the command line.

**Raises** A ValueError is raised in the case of conflicting inputs, i.e., if i) analysis_class==None and add_analysis_class_arg=False, i.e., the analysis class is not determined or ii) analysis_class!=None and add_analysis_class_arg=False, i.e, the analysis class is determined via two separate mechanisms.

**add_and_parse_analysis_arguments()**

The function assumes that the command line parser has been setup using the `initialize_argument_parser(..)`

This function is responsible for adding all command line arguments that are specific to the analysis and to then parse those argument and save the relevant data in the self.analysis_arguments dictionary. Command-line arguments that are specific to the command line driver are removed, so that only arguments that can be consumed by the analysis are handed to the analysis.

*Side effects:* The function sets `self.analysis_arguments`

**analysis_class_arg_name** = `__analysis_class`

The name where the positional argument for defining the analysis class will be stored.

**create_analysis_object()**

Initialize the analysis object, i.e., set `self.analysis_object`

**get_analysis_class_from_cl()**

Internal helper function used to get the analysis class object based on the analysis_class_arg_name positional argument from the command line.

*Side effects:* The function sets `self.analysis_class`

**Raises** ImportError in case that the analysis module cannot be loaded

**Raises** AttributeError in case that the analysis class cannot be extracted from the module

**initialize_argument_parser()**

Internal helper function used to initialize the argument parser. NOTE: `self.analysis_class` must be set before calling this function.

*Side effects:* The function sets `self.parser` and `self.required_argument_group`

**log_level_arg_name** = `loglevel`

Name of the keyword argument used to specify the level of logging to be used

**main()**

Default main function for running an analysis from the command line. The default implementation exposes all specified analysis parameters as command line options to the user. The default implementation also provides means to print a help text for the function.

** Raises** ValueError is raised in case that the analysis class is unknown

**output_save_arg_name** = `save`

Name of the key-word argument used to define

**parse_cl_arguments()**

The function assumes that the command line parser has been setup using the `initialize_argument_parser(..)`

This function parses all arguments that are specific to the command-line parser itself. Analysis arguments are added and parsed later by the `add_and_parse_analysis_arguments(...)` function. The reason for this is two-fold: i) to separate the parsing of analysis arguments and arguments of the command-line driver and ii) if the same HDF5 file is used as input and output target, then we need to open it first here in append mode before it gets opened in read mode later by the arguments.

*Side effects:* The function sets `self.output_target` and `self.profile_analysis`
print_settings()

Print the analysis settings.

remove_output_target()

This function is used to delete any output target files created by the command line driver. This is done in case that an error occurred and we do not want to leave garbage files left over.

Side effects The function modifies self.output_target

Returns Boolean indicating whether we succesfully cleaned up the output

reset_analysis_object()

Clear the analysis object and recreate it, i.e., delete self.analysis_object and set it again.

cl_workflow_driver Module Module used to help with driving the execution of analysis workflows
class omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver(workflow_executor=None,
                                                                 add_script_arg=False,
                                                                 add_output_arg=True,
                                                                 add_log_level_arg=True,
                                                                 add_profile_arg=False,
                                                                 add_mem_profile_arg=False)

Bases: omsi.workflow.driver.base.workflow_driver_base

Command-line workflow driver.

Variables

• script_arg_name – Name of the optional keyword cl argument for defining workflow scripts to be executed

• output_save_arg_name – Name of the optional keyword argument for specifying the name and target for the workflow. This may be a folder or an HDF5 file ending with .h5

• profile_arg_name – Name of the keyword argument used to enable profiling of the analysis

• profile_mem_arg_name – Name of the keyword argument used to enable profiling of memory usage of an analysis

• log_level_arg_name – Name of the keyword cl argument to define the level of logging to be used.

• workflow_executor – The workflow executor object used to execute the analysis workflow. The workflow executor must derive from omsi.workflow.executor.base.workflow_executor_base. May be None in case that we use the command-line to define workflow executor or if the default executor should be used. The default executur class is defined by omsi.workflow.executor.base.workflow_executor_base.

• script_files – List of strings with the paths to files with workflow scripts to be executed.

• add_script_arg – Boolean indicating whether the –script keyword argument should be added to the command-line, to define the workflow scripts via the CL (or whether the scripts will be set explicitly)

• add_output_arg – Boolean indicating whether an optional keyword argument should be added to define the output target for the analysis.

• add_profile_arg – Add the optional –profile keyword argument for profiling the analysis
- **add_mem_profile_arg** – Boolean indicating whether we should add the optional keyword argument for enabling memory profiling of the analysis.

- **add_log_level_arg** – Boolean indicating whether we should add the optional keyword argument to specify the logging level via the command line.

- **parser** – The argparse.ArgumentParser instance used for defining command-line arguments

- **required_argument_group** – argparse.ArgumentParser argument group used to define required command line arguments

- **optional_argument_group** – argparse.ArgumentParser argument group used to define optional command line arguments

- **custom_argument_groups** – Dict of custom argparse.ArgumentParser argument groups specified by the analysis

- **identifier_argname_separator** – String used to separate the analysis identifier and argument name when creating custom command-line options for the individual analyses of the workflow

- **output_target** – Specification of the output target where the analysis result should be stored

- **profile_analyses** – Boolean indicating whether we should profile the analysis for time and usage

- **profile_analyses_mem** – Boolean indicating whether we should profile the memory usage of the individual analyses

- **analysis_arguments** – Dictionary defining the custom input arguments to be used for the analysis

- **workflow_executor_arguments** – Dictionary defining the custom input arguments routed to the workflow executor

- **__output_target_self** – Private member variable used to store the output files created by this object.

- **user_log_level** – The custom logging level specified by the user (or None)

- **mpi_root** – The root rank used when running in parallel

- **mpi_comm** – The mpi communicator to be used when running in parallel

**Parameters**

- **workflow_executor** (oms.workflow.executor_base.workflow_executor_base) – The workflow executor object used to execute the analysis workflow. The workflow executor must derive from oms.workflow.executor_base.workflow_executor_base. May be None in case that we use the command-line to define workflow executor or if the default executor should be used. The default executor class is defined by oms.workflow.executor_base.workflow_executor_base.

- **add_script_arg** – Boolean indicating whether the –script keyword argument should be added to the command-line, to define the workflow scripts via the CL (or whether the scripts will be set explicitly)

- **add_output_arg** – Boolean indicating whether we should add the optional keyword argument for defining the output target for the analysis.

- **add_log_level_arg** – Boolean indicating whether we should add the optional keyword argument to specify the
logging level via the command line. 

:param add_profile_arg: Boolean indicating whether we should add the optional keyword argument for enabling profiling of the analysis.

**Parameters**

- **add_mem_profile_arg** – Boolean indicating whether we should add the optional keyword argument for enabling memory profiling of the analysis.

**Raises** A ValueError is raised in the case of conflicting inputs, i.e., if i) workflow_executor==None and add_script_arg=False, i.e., the analysis class is not determined or ii) workflow_executor!=None and add_script_arg=False, i.e, the analysis class is determined via two separate mechanisms.

**add_and_parse_workflow_arguments()**

The function assumes that the command line parser has been setup using the initialize_argument_parser(..).

This function is responsible for adding all command line arguments that are specific to the workflow and to then parse those arguments and save the relevant data in the self.analysis_arguments dictionary. Command-line arguments that are specific to the command line driver are removed, so that only arguments that can be consumed by the analysis are handed to the analysis.

**Side effects:** The function sets `self.analysis_arguments` and updates the analysis parameters of the analyses stored in `self.workflow_executor.analysis_tasks`.

**create_workflow_executor_object()**

Initialize the workflow executor object, i.e., set `self.workflow_executor`.

**Side effects:** This function potentially modifies `self.workflow_executor`.

**initialize_argument_parser()**

Internal helper function used to initialize the argument parser.

**Side effects:** The function sets:

- `self.parser`
- `self.required_argument_group`
- `self.opitonal_argument_group`

**log_level_arg_name = 'loglevel'**

Name of the keyword argument used to specify the level of logging to be used.

**main()**

Default main function for running an analysis from the command line. The default implementation exposes all specified analysis parameters as command line options to the user. The default implementation also provides means to print a help text for the function.

**Raises** ValueError is raised in case that the analysis class is unknown.

**output_save_arg_name = 'save'**

Name of the key-word argument used to define.

**parse_cl_arguments()**

The function assumes that the command line parser has been setup using the initialize_argument_parser(..).

This function parses all arguments that are specific to the command-line parser itself. Analysis workflow arguments are added and parsed later by the add_and_parse_workflow_arguments(...) function. The reason for this is two-fold: i) to separate the parsing of analysis arguments and arguments of the command-line driver and ii) if the same HDF5 file is used as input and output target, then we need to open it first here in append mode before it gets opened in read mode later by the arguments.

**Side effects:** The function sets:
• self.output_target
• self.profile_analyses

print_memory_profiles()
Print the memory profiles if available

print_settings()
Print the analysis settings.

print_time_and_usage_profiles()
Print the profiling data for time and usage if available

profile_arg_name = 'profile'
Name of the keyword argument used to enable profiling of the analysis

profile_mem_arg_name = 'memprofile'
Name of the keyword argument used to enable profiling of memory usage of an analysis

remove_output_target()
This function is used to delete any output target files created by the command line driver. This is done in case that an error occurred and we do not want to leave garbage files left over.

Side effects
The function modifies self.output_target

Returns
Boolean indicating whether we successfully cleaned up the output

reset_workflow_executor_object()
Remove and recreate the workflow executor object

script_arg_name = 'script'
The name where the positional argument for defining the analysis class will be stored.

executor Package

executor Package Package with executors of analysis workflows. Executors perform and control the execution of workflows. The setup of workflows is often performed by the user or via dedicated workflow drivers.

base Module Module containing base classes for workflow executors.

Workflow executors control the execution of workflows. The setup of workflows is often controlled either by a workflow driver or the user.

class omsi.workflow.executor.base.workflow_executor_base(analysis_objects=None)
Bases: omsi.datastructures.analysis_data.parameter_manager

Base class used to execute a workflow of one or many analyses. This is an object-based execution, i.e., the user defines a set of analyses to be executed.

We are given a set of existing analysis objects for which we need to coordinate the execution.

Variables

• analysis_objects – Private set of analysis objects to be executed
  ...
• DEFAULT_EXECUTOR_CLASS – Define the derived workflow_executor_base class to be used as default executor. The default value is None, in which case the greedy_workflow_executor is used. This variable is used by the get_default_executor function to instantiate a default workflow executor on request. Using this variable we can change the default executor to our own preferred executor, e.g., to change the executor used by the omsi.analysis.base.analysis_base functions execute_all(...) and execute_recursive(...)

10.1. Subpackages 215
To implement a derived workflow executor, we need to implement a derived class that implements the main() function.

Initialize the workflow executor

**Parameters** `analysis_objects` – A list of analysis objects to be executed

**DEFAULT_EXECUTOR_CLASS** = None

The default executor class to be used

**add_analysis** (`analysis_object`)

Add a given analysis to the set of object to be executed by the workflow

Shorthand for: `self.analysis_tasks.add(analysis_object)`

**add_analysis_all** ()

Add all known analyses to the workflow.

Shorthand for: `self.analysis_tasks.add_analysis_all()`

**add_analysis_dependencies** ()

Add the dependencies of all analyses to the workflow in case they are missing.

This function is recursive, step-by-step adding all dependencies of the workflow to the list of tasks to be executed, until no more dependencies are found.

Usually this function is called by the workflow executor itself before running the analysis and should not need to be called by the user.

**Returns** Integer indicating the number of dependencies added to the list of tasks

**add_analysis_from_scripts** (`script_files`)

Evaluate the list of scripts and add all (i.e., zero, one, or multiple) analyses to this workflow

**NOTE:** This function executes scripts using exec(..), i.e., there are NO safeguards against malicious codes.

**Parameters** `script_files` – List of strings with the paths to the script files. If only a single script is used, then a single string may be used as well.

**clear** ()

Remove all analyses from the workflow.

Shorthand for: `self.analysis_tasks.clear()`

**execute** ()

Execute the workflow. This uses the main() function to run the actual workflow.

**classmethod from_script_files** (`script_files`)

Same as from_script, but the scripts are read from the given list of files.

**NOTE:** This function executes scripts using exec(..), i.e., there are NO safeguards against malicious codes.

**Parameters** `script_files` – List of strings with the paths to the script files. If only a single script is used, then a single string may be used as well.

**Returns** Instance of the current workflow executor class for running the given workflow

**classmethod from_scripts** (`scripts`)

Create and initialize a workflow executor of the current class type to execute the workflow defined in the given set of scripts.

This function using `analysis_task_list.from_scripts` to evaluate the workflow scripts to extract all analyses to be created.

**NOTE:** This function executes scripts using exec(..), i.e., there are NO safeguards against malicious codes.
Parameters `scripts` – The script with the setup of the workflow. This should only include the definition of analyses and their inputs.

**Returns** Instance of the current workflow executor class for running the given workflow

```python
def get_analyses()
    Get the list of analyses to be run.
    Shorthand for: self.analysis_tasks
```

```python
def get_analysis(index)
    Get the analysis with the given index
    Shorthand for: self.analysis_tasks[index]
```

**Parameters**
- `index` – Integer index of the analysis

**Returns** omsi.analysis.base.analysis_base object

**Raises** IndexError in case that the index is invalid

```python
def get_default_executor(analysis_objects=None)
    Create an instance of the default workflow executor to be used.
    Parameters
    - `analysis_objects` – A set or unique list of analysis objects to be executed by the workflow
    **Returns** Instance of the default workflow executor
```

```python
def get_default_executor_class()
    Get the default executor class
    **Returns** Derived class of workflow_executor_base
```

```python
def main()
    Implement the execution of the workflow. We should always call execute(..) or __call__(..) to run the workflow. This function is intended to implementd the executor-specific execution behavior and must be implemented in child classes.
```

**greedy_executor Module** Module used to help with the execution of complex analyses workflows

```python
class omsi.workflow.executor.greedy_executor.greedy_executor(analysis_objects=None)
    Bases: omsi.workflow.executor.base.workflow_executor_base
    Execute a set of analysis objects and their dependencies
```

**Variables**
- `run_info` – The runtime information dictionary for the overall workflow
- `mpi_comm` – The MPI communicator to be used when running in parallel
- `mpi_root` – The MPI root rank when running in parallel

**Additional parameters:**

**Parameters**
- `reduce_memory_usage` – Boolean indicating whether we should reduce memory usage by pushing analysis data to file after an analysis has been completed. This reduces the amount of data we keep in memory but results in additional overhead for I/O and temporary disk storage.

Initialize the workflow driver

**Parameters**
- `analysis_objects` – A list of analysis objects to be executed
main()  
   Execute the analysis workflow

10.1.6 tools Package

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>omsi.tools</code></td>
<td>Package for collecting tools (e.g.,</td>
</tr>
<tr>
<td><code>omsi.tools.convertToOMSI</code></td>
<td>Tool used to convert img files to OpenMSI HDF5 files.</td>
</tr>
<tr>
<td><code>omsi.tools.run_analysis</code></td>
<td>Simple helper tool to run an analysis.</td>
</tr>
<tr>
<td><code>omsi.tools.run_workflow</code></td>
<td>Simple helper tool to run a workflow.</td>
</tr>
<tr>
<td><code>omsi.tools.misc</code></td>
<td>Collection of miscellaneous tools.</td>
</tr>
<tr>
<td><code>omsi.tools.misc.create_peak_cube_overview</code></td>
<td>Simple helper tool used to generate a set of PNG images for a</td>
</tr>
<tr>
<td></td>
<td>global peak analysis (one per global peak) as well as a LaTeX</td>
</tr>
<tr>
<td><code>omsi.tools.misc.make_thumb</code></td>
<td>document that summarizes all the images in a single document.</td>
</tr>
<tr>
<td><code>omsi.tools.experimental</code></td>
<td>Collection of experimental tools and tools under development.</td>
</tr>
</tbody>
</table>

convertToOMSI Module

Tool used to convert img files to OpenMSI HDF5 files.
For usage information execute: python convertToOMSI –help

class `omsi.tools.convertToOMSI.ConvertFiles`

   Bases: object

   Class providing a number of functions for converting various file types to OMSI, including a number of helper functions related to the data conversion.

   static `check_format(name, format_type)`
       Helper function used to determine the file format that should be used

       Parameters

       - `name` – Name of the folder/file that we should read
       - `format_type` – String indicating the format-option given by the user. If the format is not determined (i.e., “auto”) then this function tries to determine the appropriate format. Otherwise this option is returned as is, as the user explicitly said which format should be used.

       Returns String indicating the appropriate format. Returns None in case no valid option was found.

   static `convert_files()`
       Convert all files in the given list of files with their appropriate conversion options

   static `create_dataset_list(input_filenames, format_type=None, data_region_option='split+merge')`
       Based on the list of input_filenames, generate the ConvertSettings.dataset_list, which contains a dictionary describing each conversion job

   Parameters

       - `input_filenames` – List of names of files to be converted.
       - `format_type` – Define which file-format should be used. Default value is ‘auto’ indicating the function should determine for each file the format to be used. See also ConvertSettings.available_formats parameter.
• **data_region_option** – Define how different regions defined for a file should be handled. E.g., one may want to split all regions into individual datasets (‘split’), merge all regions into a single dataset (‘merge’), or do both (‘split+merge’). See also the ConvertSettings.available_region_options parameter for details. By default the function will do ‘split+merge’.

**Returns**

List of dictionaries describing the various conversion jobs. Each job is described by a dict with the following keys:

- ‘basename’ : The base name of the file or directory with the data
- ‘format’ : The data format to be used
- ‘dataset’ : The index of the dataset to be converted if the input stores multiple data cubes
- ‘region’ : The index of the region to be converted if the input defines multiple regions
- ‘exp’ : One of ‘previous’ or ‘new’, defining whether a new experiment should be created or whether the experiment from the previous conversion(s) should be reused.

**static suggest_chunking** (xsize, ysize, mzsize, dtype, print_results=False)

Helper function used to suggest good chunking strategies for a given data cube

**Parameters**

- **xsize** – Size of the dataset in x.
- **ysize** – Size of the dataset in y.
- **mzsize** – Size of the dataset in mz.
- **print_results** – Print the results to the console.

**Returns**

Three tuples:

- **spectrum_chunk** : The chunking to be used to optimize selection of spectra.
- **slice_chunk** : The chunking to be used to optimize selection of image slices.
- **balanced_chunk** : The chunking that would provide a good balance in performance for different selection strategies.

**static suggest_chunkings_for_files** (in_dataset_list)

Helper function used to suggest good chunking strategies for a given set of files.

**Parameters** **in_dataset_list** – Python list of dictionaries describing the settings to be used for the file conversion

**Returns** This function simply prints results to standard-out but does not return anything.

**static write_data** (input_file, data, data_io_option='spectrum', chunk_shape=None, write_progress=True)

Helper function used to implement different data write options.

**Parameters**

- **input_file** – The input data file
- **data** – The output dataset (either an h5py dataset or omsi_file_msidata object.
- **data_io_option** – String indicating the data write method to be used. One of:
  - **spectrum**: Write the data one spectrum at a time
- **all**: Write the complete dataset at once.
- **chunk**: Write the data one chunk at a time.
- **chunk_shape**: The chunking used by the data. Needed to decide how the data should be written when a chunk-aligned write is requested.
- **write_progress** *(bool)*: Write progress in % to standard out while data is being written.

class omsi.tools.convertToOMSI.ConvertSettings
Bases: object

This class is used specify the settings for the data conversion

```python
add_file_to_db = True
auto_chunk = True
available_error_options = ['terminate-and-cleanup', 'terminate-only', 'continue-on-error']
available Formats = {'imzml_file': <class 'omsi.dataformat.imzml_file.imzml_file'>, 'bruckerflex_file': <class 'omsi.dataformat.bruckerflex_file.bruckerflex_file'>
available io_options = ['chunk', 'spectrum', 'all']
available region_options = ['split', 'merge', 'split+merge']
check_add_nersc = True
chunks = (4, 4, 2048)
compression = 'gzip'
compression_opts = 4
dataset_list = []
```

**Parameters** **dataList** – List of python dictionaries describing specific conversion settings for each conversion task. Each dictionary contains the following keys:

- **basename**: Name of the file to be converted
- **format**: File format to be used (see ConvertSettings.available Formats)
- **exp**: Indicate the experiment the dataset should be stored with. Valid values are
  - **new**: Generate a new experiment for the dataset
  - **previous**: Use the same experiment as used for the previous dataset
  - 1, 2, 3... : Integer value indicating the index of the experiment to be used.
- **region**: Optional key with index of the region to be converted. None to merge all regions.
- **dataset**: Optional key with index of the dataset to be converted.
- **omsi_object**: Optional key used to save a pointer to the omsi data object with the converted data
- **dependencies**: Additional dependencies that should be added for the dataset

db_server_url = 'https://openmsi.nersc.gov/'
email_error_recipients = []
email_success_recipients = []
error_handling = 'terminate-and-cleanup'
execute_fpg = True
execute_fpl = False
execute_nmf = True
execute_ticnorm = False
file_user = ‘oruebel’
format_option = None
generate_thumbnail = False
generate_xdmf = False
io_block_size_limit = 524288000
io_option = ‘spectrum_to_image’
job_id = None
metadata = {}
nmf_num_component = 20
nmf_num_iter = 2000
nmf_timeout = 600
nmf_tolerance = 0.0001
nmf_use_raw_data = False
omsi_output_file = None

classmethod parse_input_args (argv)
Process input parameters and define the script settings.

Parameters argv – The list of input arguments

Returns
This function returns the following four values:

• ‘input_error’ : Boolean indicating whether an error has occurred during the processing of
the inputs
• ‘inputWarning’ : Boolean indicating whether a warning occurred during the processing of
the inputs
• ‘output_filename’ : Name for the output HDF5 file
• ‘input_filenames’ : List of strings indicating the list of input filenames

classmethod print_help()
Function used to print the help for this script

recorded_warnings = []
region_option = ‘split+merge’
suggest_file_chunkings = False
user_additional_chunks = []

omsi.tools.convertToOMSI.main (argv=None)
The main function defining the control flow for the conversion
run_analysis Module

Simple helper tool to run an analysis. This is essentially just a short-cut to the omsi/workflow/analysis_driver/omsi_cl_diver module

run_workflow Module

Simple helper tool to run an analysis. This is essentially just a short-cut to the omsi/workflow/analysis_driver/omsi_cl_diver module

Subpackages

tools.experimental Package

tools.experimental Package  Collection of experimental tools and tools under development.

tools.misc Package

tools.misc Package  Collection of miscellaneous tools.

make_thumb Module  Simple script to generate thumbnail images

omsi.tools.misc.make_thumb.main(argv=None)
Then main function

create_peak_cube_overview Module  Simple helper tool used to generate a set of PNG images for a global peak analysis (one per global peak) as well as a LaTeX document that summarizes all the images in a single document.

NOTE: The module will try to build the LaTeX document using pdflatex, i.e., it is assumed that pdflatex is available.

omsi.tools.misc.create_peak_cube_overview.main(argv=None)
Then main function
omsi.tools.misc.create_peak_cube_overview.print_help()
Print the user help information to standard out.

10.1.7 examples Package

omsi.examples Package with a collection of various misc.

simple_viewer Module

Simple viewer for OpenMSI data

class omsi.examples.simple_viewer.MyViewer(data, mzdata)
Create a simple viewer with image of the data and a curve plot for a spectrum.

Parameters

• data – Reference to the hdf5 dataset of the image
• **mzdata** – mz values of the instrument to be displayed as axis in the curve plot. may be
None in case the mz data is unknown

```python
cmsi.examples.simple_viewer.main(argv=None)
```
Then main function

**testHDF5Optimization Module**

Simple test script used to test the performance of different HDF5 optimizations (using chunking) to improve the performance of hyperslap selections

```python
cmsi.examples.testHDF5Optimization.generateTestFile(omsiOutFile, xdim, ydim, zdim, xchunk, ychunk, zchunk)
cmsi.examples.testHDF5Optimization.main(argv=None)
```
Then main function

```python
cmsi.examples.testHDF5Optimization.printHelp()
```
Print the help explaining the usage of testHDF5Optimization

**testHDF5Optimization_alignedWrite Module**

```python
cmsi.examples.testHDF5Optimization_alignedWrite.generateTestFile(omsiOutFile, xdim, ydim, zdim, xchunk, ychunk, zchunk)
cmsi.examples.testHDF5Optimization_alignedWrite.main(argv=None)
```
Then main function

```python
cmsi.examples.testHDF5Optimization_alignedWrite.printHelp()
```
Print the help explaining the usage of testHDF5Optimization

**test_multiprocess_slice Module**

```python
cmsi.examples.test_multiprocess_slice.create_process(argv)
cmsi.examples.test_multiprocess_slice.generateBaseTestFile(omsiOutFile, xdim, ydim, zdim)
cmsi.examples.test_multiprocess_slice.generateChunkedTestFile(omsiOutFile, xdim, ydim, zdim, xchunk, ychunk, zchunk, compress=False, donorFile='/project/projectdirs/openmsi/omsi_data/old/TEST.h5')
cmsi.examples.test_multiprocess_slice.main(argv=None)
```
Then main function

```python
cmsi.examples.test_multiprocess_slice.sliceSelect(args)
cmsi.examples.test_multiprocess_slice.spectraSelect(args)
```
test_par Module

cmsi.examples.test_par.create_process(argv)
cmsi.examples.test_par.generateBaseTestFile(omsiOutFile, xdim, ydim, zdim)
cmsi.examples.test_par.generateChunkedTestFile(omsiOutFile, xdim, ydim, zdim, xchunk, ychunk, zchunk, compress=False, donorFile='/project/projectdirs/openmsi/omsi_data/old/TEST.h5')
cmsi.examples.test_par.main(argv=None)
   Then main function
cmsi.examples.test_par.sliceSelect(args)
cmsi.examples.test_par.spectraSelect(args)

test_peakcube Module

cmsi.examples.test_peakcube.main(argv=None)
   Then main function
cmsi.examples.test_peakcube.printHelp()

testhdf5_file_read Module

Simple test script used to test the performance of different HDF5 optimizations (using chunking) to improve the performance of hyperslap selections
cmsi.examples.testhdf5_file_read.main(argv=None)
   Then main function
cmsi.examples.testhdf5_file_read.printHelp()
   Print the help explaining the usage of testHDF5Optimiation

10.1.8 templates Package

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omsi.templates</td>
<td>This package provides a collection of code templates to ease the development of additional components, e.g., analysis modules. As such, this package is NOT intended for direct usage but is rather just a library of code templates.</td>
</tr>
<tr>
<td>omsi.templates.analysis_template</td>
<td>Template intended to help with the development of new analysis modules.</td>
</tr>
</tbody>
</table>

templates Package

This package provides a collection of code templates to ease the development of additional components, e.g., analysis modules. As such, this package is NOT intended for direct usage but is rather just a library of code templates.

analysis_template Module

Template intended to help with the development of new analysis modules.

class omsi.templates.analysis_template.analysis_template(name_key='undefined')
   Bases: omsi.analysis.base.analysis_base
      Template intended to help with the development of new analysis classes.
      Search for EDIT_ME to find locations that need to be changed.
EDIT_ME Search for analysis_template and replace it with your classname throughout

EDIT_ME Replace this doc-string with your class documentation

Initialize the basic data members

**execute_analysis**()

EDIT_ME Implement this function to implement the execution of the actual analysis.

This function may not require any input parameters. All input parameters are recoded in the parameters and dependencies lists and should be retrieved from there, e.g., using basic slicing `self[paramName]`

EDIT_ME Remove this comment and replace it with your own documentation. Describe what your analysis does and how a user can use it. Note, a user will call the function `execute(...) which takes care of storing parameters, collecting execution data etc., so that you only need to implement your analysis, the rest is taken care of by `analysis_base`.

Keyword Arguments:

**Parameters**

- `mydata` — ...

**Returns** This function may return any developer-defined data. Note, all output that should be recorded must be put into the data list.

**groups** = None

EDIT_ME

Add a list of names of input parameters for your analysis using the `self.add_add_parameter`.

For parameters that define n-dimension arrays you should specify the `dtypes['ndarray']` as dtype. This will allow discovery of dependencies and proper function with the analysis drivers. Other available dtypes include standard built-ins, e.g., int, float etc. See `self.get_default_dtypes()` for details.

It is also recommended to organize parameters into groups using the default set of groups, e.g., groups['input'], groups['settings'], groups['stop'].

**classmethod** `v_qmz` (**analysis_object**, `qslice_viewer_option=0`, `qspectrum_viewer_option=0`)

Get the mz axes for the analysis

**Parameters**

- `analysis_object` – The omsi_file_analysis object for which slicing should be performed
- `qslice_viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qslice URL pattern.
- `qspectrum_viewer_option` – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them for the qspectrum URL pattern.

**Returns**

The following four arrays are returned by the analysis:

- `mz_spectra` : Array with the static mz values for the spectra.
- `label_spectra` : Label for the spectral mz axis
- `mz_slice` : Array of the static mz values for the slices or None if identical to the `mz_spectra`.
- `label_slice` : Lable for the slice mz axis or None if identical to `label_spectra`.

**classmethod** `v_qslice` (**analysis_object**, `z`, `viewer_option=0`)

Get 3D analysis dataset for which z-slices should be extracted for presentation in the OMSI viewer
Parameters

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed
- **z** – Selection string indicating which z values should be selected.
- **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

numpy array with the data to be displayed in the image slice viewer. Slicing will be performed typically like [:,:zmin:zmax].

```python
classmethod v_qslice_viewer_options(analysis_object)
```

Get a list of strings describing the different default viewer options for the analysis for qslice. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qslice should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

**Parameters**

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

**Returns**

List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qslice requests (i.e., v_qslice(...) is not available).

```python
classmethod v_qspectrum(analysis_object, x, y, viewer_option=0)
```

Get from which 3D analysis spectra in x/y should be extracted for presentation in the OMSI viewer

**Developer Note:** h5py currently supports only a single index list. If the user provides an index-list for both x and y, then we need to construct the proper merged list and load the data manually, or if the data is small enough, one can load the full data into a numpy array which supports multiple lists in the selection.

**Parameters**

- **analysis_object** – The omsi_file_analysis object for which slicing should be performed
- **x** – x selection string
- **y** – y selection string
- **viewer_option** – If multiple default viewer behaviors are available for a given analysis then this option is used to switch between them.

**Returns**

The following two elements are expected to be returned by this function:

1. 1D, 2D or 3D numpy array of the requested spectra. NOTE: The mass (m/z) axis must be the last axis. For index selection x=1,y=1 a 1D array is usually expected. For indexList selections x=[0]&y=[1] usually a 2D array is expected. For range selections x=0:1&y=1:2 we one usually expects a 3D array.

2. None in case that the spectra axis returned by v_qmz are valid for the returned spectrum. Otherwise, return a 1D numpy array with the m/z values for the spectrum (i.e., if custom m/z values are needed for interpretation of the returned spectrum). This may be needed, e.g., in cases where a per-spectrum peak analysis is performed and the peaks for each spectrum appear at different m/z values.
classmethod v_qspectrum_viewer_options (analysis_object)

Get a list of strings describing the different default viewer options for the analysis for qspectrum. The default implementation tries to take care of handling the spectra retrieval for all the dependencies but can naturally not decide how the qspectrum should be handled by a derived class. However, this implementation is often called at the end of custom implementations to also allow access to data from other dependencies.

param analysis_object The omsi_file_analysis object for which slicing should be performed. For most cases this is not needed here as the support for slice operations is usually a static decision based on the class type, however, in some cases additional checks may be needed (e.g., ensure that the required data is available).

returns List of strings indicating the different available viewer options. The list should be empty if the analysis does not support qspectrum requests (i.e., v_qspectrum(...) is not available).
omsi.analysis, 78
omsi.analysis.analysis_views, 108
omsi.analysis.base, 96
omsi.analysis.compound_stats, 130
omsi.analysis.compound_stats.experimental, 131
omsi.analysis.compound_stats.omsi_score_midas, 130
omsi.analysis.compound_stats.omsi_score_pactolus, 131
omsi.analysis.compound_stats.third_party, 131
omsi.analysis.findpeaks, 110
omsi.analysis.findpeaks.experimental, 113
omsi.analysis.findpeaks.experimental.mypeakfinder, 117
omsi.analysis.findpeaks.experimental.omsi_lpf, 114
omsi.analysis.findpeaks.experimental.omsi_npg, 116
omsi.analysis.findpeaks.experimental.omsi_peakcube, 114
omsi.analysis.findpeaks.experimental.pfrun, 114
omsi.analysis.findpeaks.omsi_findpeaks_global, 112
omsi.analysis.findpeaks.omsi_findpeaks_local, 112
omsi.analysis.findpeaks.third_party, 113
omsi.analysis.findpeaks.third_party.findpeaks, 113
omsi.analysis.generic, 106
omsi.analysis.msi_filtering, 121
omsi.analysis.msi_filtering.experimental, 126
omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask, 126
omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster, 128
omsi.analysis.msi_filtering.omsi_tic_norm, 124
omsi.analysis.msi_filtering.third_party, 126
omsi.analysis.multivariate_stats, 117
omsi.analysis.multivariate_stats.experimental, 121
omsi.analysis.multivariate_stats.omsi_nmf, 120
omsi.analysis.multivariate_stats.third_party, 120
omsi.analysis.multivariate_stats.third_party.nmf, 121
omsi.dataformat.bruckerflex_file, 172
omsi.dataformat.file_reader_base, 165
omsi.dataformat.img_file, 169
omsi.dataformat.mzml_file, 170
omsi.dataformat.omsi_file, 133
omsi.dataformat.omsi_file.analysis, 154
omsi.dataformat.omsi_file.common, 137
omsi.dataformat.omsi_file.dependencies, 158
omsi.dataformat.omsi_file.experiment, 141
omsi.dataformat.omsi_file.format, 133
omsi.dataformat.omsi_file.instrument, 145
omsi.dataformat.omsi_file.main_file, 140
omsi.dataformat.omsi_file.metadata_collection, 143
omsi.dataformat.omsi_file.methods, 147
omsi.dataformat.omsi_file.msidata, 148
omsi.datastructures, 176
omsi.datastructures.analysis_data, 176
omsi.datastructures.dependency_data, 180
omsi.datastructures.metadata, 184
omsi.datastructures.metadata.metadata_data, 184
omsi.datastructures.metadata.metadata_ontologies, 185
omsi.datastructures.run_info_data, 182
omsi.examples.simple_viewer, 222
omsi.examples.test_multiprocess_slice, 223
omsi.examples.test_par, 224
omsi.examples.test_peakcube, 224
omsi.examples.testhdf5_file_read, 224
omsi.examples.testHDF5Optimization, 223
omsi.examples.testHDF5Optimization_alignedWrite, 223
omsi.shared.data_selection, 186
omsi.shared.log, 199
omsi.shared.mpi_helper, 203
omsi.shared.omsi_web_helper, 196
omsi.shared.spectrum_layout, 198
omsi.templates, 224
omsi.templates.analysis_template, 224
omsi.tools.convertToOMSI, 218
omsi.tools.experimental, 222
omsi.tools.misc, 222
omsi.tools.misc.create_peak_cube_overview, 222
omsi.tools.misc.make_thumb, 222
omsi.tools.run_analysis, 222
omsi.workflow, 206
omsi.workflow.common, 206
omsi.workflow.driver, 208
omsi.workflow.driver.base, 209
omsi.workflow.driver.cl_analysis_driver, 209
omsi.workflow.driver.cl_workflow_driver, 212
omsi.workflow.executor, 215
omsi.workflow.executor.base, 215
omsi.workflow.executor.greedy_executor, 217
Symbols

_mzml_file__compute_coordinates() (omsi.dataformat.mzml_file.mzml_file method), 170
_mzml_file__compute_filetype() (omsi.dataformat.mzml_file.mzml_file class method), 170
_mzml_file__compute_mz_axis() (omsi.dataformat.mzml_file.mzml_file class method), 170
_mzml_file__compute_num_scans() (omsi.dataformat.mzml_file.mzml_file static method), 170
_mzml_file__compute_scan_dependencies() (omsi.dataformat.mzml_file.mzml_file static method), 170
_mzml_file__compute_scan_types_and_indices() (omsi.dataformat.mzml_file.mzml_file method), 170
_mzml_file__parse_scan_parameters() (omsi.dataformat.mzml_file.mzml_file static method), 171
_mzml_file__read_all() (omsi.dataformat.mzml_file.mzml_file method), 171
_omsi_file_dependencies__create_dependency_graph_node() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencies static method), 161

A

add() (omsi.workflow.common.analysis_task_list method), 206
add_all() (omsi.workflow.common.analysis_task_list method), 206
add_analysis() (omsi.workflow.executor.base.workflow_executor_base method), 216
add_analysis_all() (omsi.workflow.executor.base.workflow_executor_base method), 216
add_analysis_dependencies() (omsi.workflow.common.analysis_task_list method), 206
add_analysis_dependencies() (omsi.workflow.executor.base.workflow_executor_base method), 216
add_analysis_from_scripts() (omsi.workflow.executor.base.workflow_executor_base method), 216
add_and_parse_analysis_arguments() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
add_and_parse_workflow_arguments() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 214
add_custom_data_to_omsi_file() (omsi.analysis.analysis_base method), 81
add_custom_data_to_omsi_file() (omsi.analysis.base.analysis_base method), 98
add_dependency() (omsi.dataformat.omsi_file_dependencies.oms_dependencies_manager method), 158
add_dependency() (omsi.dataformat.omsi_file_dependencies.oms_dependencies_manager method), 161
add_file_to_db (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
add_metadata() (omsi.dataformat.omsi_file代谢.collection.oms_file_代谢.collection method), 143
add_parameter() (omsi.analysis.analysis_base method), 81
add_parameter() (omsi.analysis.base.analysis_base method), 99
add_parameter() (omsi.datastructures.analysis_data.parameter_manager method), 179
all() (omsi.workflow.common.analysis_task_list class method), 206
allowed_nersc_locations (omsi.shared.omsi_web_helper.WebHelper attribute), 197
ana_hdf5link (omsi.analysis.analysis_data attribute), 79
ana_hdf5link (omsi.datastructures.analysis_data-analysis_data attribute), 177
analysis_base (class in omsi.analysis), 79
analysis_base (class in omsi.analysis.base), 96
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

analysis_class_arg_name (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver attribute), 211
analysis_data (class in omsi.analysis), 78
analysis_data (class in omsi.datastructures.analysis_data), 176
analysis_driver_base (class in omsi.workflow.driver.base), 209
analysis_generic (class in omsi.analysis), 89
analysis_generic (class in omsi.analysis.generic), 106
analysis_groupname (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
analysis_identifier (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
analysis_identifier_defined() (omsi.analysis.analysis_base method), 81
analysis_identifier_defined() (omsi.analysis.base.analysis_base method), 99
analysis_identifiers_unique() (omsi.workflow.common.analysis_task_list method), 206
analysis_name_to_class() (omsi.analysis.analysis_views.analysis_views class method), 108
analysis_parameter_group (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
analysis_parameter_help_attr (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
analysis_runinfo_group (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
analysis_task_list (class in omsi.workflow.common), 206
analysis_template (class in omsi.templates.analysis_template), 224
analysis_type (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
analysis_views (class in omsi.analysis.analysis_views), 108
AnalysisReadyError, 89, 96
append() (omsi.workflow.common.analysis_task_list method), 206
auto_chunk (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
available_analysis() (omsi.analysis.analysis_views.analysis_views class method), 108
available_analysis_descriptions() (omsi.analysis.analysis_views.analysis_views class method), 108
available_error_options (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
available_formats (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
available_formats() (omsi.dataformat.file_reader_base.file_reader_base static method), 166
available_io_options (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
available_mzml_types (omsi.dataformat.mzml_file.mzml_file attribute), 171
available_region_options (omsi.tools.convertToOMSI.ConvertSettings attribute), 220

B
barrier() (in module omsi.shared.mpi_helper), 203
bastet_analysis() (in module omsi.analysis.generic), 108
bool_type() (omsi.datastructures.analysis_data.data_dtypes static method), 177
broadcast() (in module omsi.shared.mpi_helper), 203
bruckerflex_file (class in omsi.dataformat.bruckerflex_file), 172

C
check_add_nersc (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
check_format() (omsi.tools.convertToOMSI.ConvertFiles static method), 218
check_ready_to_execute() (omsi.analysis.analysis_base method), 81
check_ready_to_execute() (omsi.analysis.base.analysis_base method), 99
check_selection_string() (in module omsi.shared.data_selection), 187
chunks (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
cl_analysis_driver (class in omsi.workflow.driver.cl_analysis_driver), 209
cl_peakfind() (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 114
cl_workflow_driver (class in omsi.workflow.driver.cl_workflow_driver), 212
clean_up() (omsi.datastructures.run_info_data.run_info_dict method), 182
clear() (omsi.datastructures.run_info_data.run_info_dict method), 182
clear() (omsi.workflow.common.analysis_task_list method), 207
clear() (omsi.workflow.executor.base.workflow_executor_base method), 216
clear_analysis() (omsi.analysis.analysis_base method), 81
clear_analysis() (omsi.analysis.base.analysis_base method), 99
clear_analysis_data() (omsi.analysis.analysis_base method), 82
clear_analysis_data() (omsi.analysis.base.analysis_base method), 99
clear_and_restore() (omsi.analysis.analysis_base method), 82
clear_and_restore() (omsi.analysis.base.analysis_base method), 99
clear_data() (omsi.datastructures.analysis_data.parameter_data method), 178
clear_parameter_data() (omsi.analysis.analysis_base method), 82
clear_parameter_data() (omsi.analysis.base.analysis_base method), 100
clear_parameter_data() (omsi.datastructures.analysis_data.parameter_manager method), 179
clear_run_info_data() (omsi.analysis.analysis_base method), 82
clear_run_info_data() (omsi.analysis.base.analysis_base method), 100
close_file() (omsi.dataformat.bruckerflex_file.bruckerflex_file method), 174
close_file() (omsi.dataformat.file_reader_base.file_reader_base method), 166
close_file() (omsi.dataformat.img_file.img_file method), 169
close_file() (omsi.dataformat.mzml_file.mzml_file method), 171
close_file() (omsi.dataformat.omsi_file.main_file.omsi_file method), 141
collect_data() (omsi.shared.mpi_helper.parallel_over_axes method), 205
comp_lev_exact() (omsi.analysis.multivariate_stats.omsi_cx class method), 118
copy() (omsi.datastructures.analysis_data.parameter_data method), 178
copy() (omsi.datastructures.dependency_data.dependency_dict method), 181
copy_dataset() (omsi.dataformat.omsi_file.msidata.omsi_file_msidata method), 150
create_analysis() (omsi.dataformat.omsi_file.analysis.omsi_analysis_manager method), 155
create_analysis_object() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
create_analysis_static() (omsi.dataformat.omsi_file.analysis.omsi_analysis_manager static method), 155
create_dataset_list() (omsi.tools.convertToOMSI.ConvertFiles static method), 218
create_dependencies() (omsi.dataformat.omsi_file.dependencies.omsi_dependencies_manager method), 158
create_experiment() (omsi.dataformat.omsi_file_experiment.omsi_experiment_manager method), 142
create_instrument_info() (omsi.dataformat.omsi_file.instrument.omsi_instrument_manager method), 146
create_metadata_collection() (omsi.dataformat.omsi_file.metadata_collection.omsi_metadata_collection_manager method), 144
create_method_info() (omsi.dataformat.omsi_file.methods.omsi_method_manager method), 147
create_msidata_full_cube() (omsi.dataformat.omsi_file.msidata.omsi_msidata_manager method), 151
create_msidata_partial_cube() (omsi.dataformat.omsi_file.msidata.omsi_msidata_manager method), 152
create_msidata_partial_spectra() (omsi.dataformat.omsi_file.msidata.omsi_msidata_manager method), 153
create_optimized_chunking() (omsi.dataformat.omsi_file.msidata.omsi_file_msidata method), 150
create_path_string() (omsi.dataformat.omsi_file.common.omsi_file_common static method), 137
create_process() (in module omsi.examples.test_multiprocess_slice), 223
create_process() (in module omsi.examples.test_par), 224
create_workflow_executor_object() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 214
critical() (omsi.shared.log.log_helper class method), 199
current_version (omsi.dataformat.omsi_file.format.omsi_format_analysis attribute), 133
current_version (omsi.dataformat.omsi_file.format.omsi_format_common attribute), 133
current_version (omsi.dataformat.omsi_file.format.omsi_format_data attribute), 134
current_version (omsi.dataformat.omsi_file.format.omsi_format_dependencies attribute), 134
current_version (omsi.dataformat.omsi_file.format.omsi_format_dependencydata attribute), 134
current_version (omsi.dataformat.omsi_file.format.omsi_format_experiment attribute), 135
current_version (omsi.dataformat.omsi_file.format.omsi_format_file attribute), 135
current_version (omsi.dataformat.omsi_file.format.omsi_format_instrument attribute), 135
current_version (omsi.dataformat.omsi_file.format.omsi_format_metadata_collection attribute), 136
current_version (omsi.dataformat.omsi_file.format.omsi_format_methods attribute), 136
current_version (omsi.dataformat.omsi_file.format.omsi_format_msidata attribute), 136

data_dtypes (class in omsi.datastructures.analysis_data), 177
data_groupname (omsi.dataformat.omsi_file.format.omsi_format_data attribute), 134
data_ready() (omsi.datastructures.analysis_data.parameter_data method), 178
data_set() (omsi.datastructures.analysis_data.parameter_data method), 178
dataset_list (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
dataset_name (omsi.dataformat.omsi_file.format.omsi_format_data attribute), 134
db_server_url (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
debug() (omsi.shared.log.log_helper class method), 200
default_db_server_url (omsi.shared.omsi_web_helper.WebHelper attribute), 197
DEFAULT_EXECUTOR_CLASS (omsi.workflow.executor.base.workflow_executor_base attribute), 216
default_keys (omsi.datastructures.analysis_data.parameter_data attribute), 178
DEFAULT_OUTPUT_PREFIX (omsi.analysis.analysis_generic attribute), 89
DEFAULT_OUTPUT_PREFIX (omsi.analysis.generic.analysis_generic attribute), 107
DEFAULT_TIME_FORMAT (omsi.datastructures.run_info_data.run_info_dict attribute), 182
define_missing_parameters() (omsi.analysis.analysis_base method), 82
define_missing_parameters() (omsi.analysis.base.analysis_base method), 100
define_missing_parameters() (omsi.datastructures.analysis_data.parameter_manager method), 179
dependencies_groupname (omsi.dataformat.omsi_file.format.omsi_format_dependencies attribute), 134
dependency_datasetname (omsi.dataformat.omsi_file.format.omsi_format_dependencydata attribute), 134
dependency_dict (class in omsi.datastructures.dependency_data), 180
dependency_mainname (omsi.dataformat.omsi_file.format.omsi_format_dependencydata attribute), 134
dependency_parameter (omsi.dataformat.omsi_file.format.omsi_format_dependencydata attribute), 134
dependency_parameter_help_attr (omsi.dataformat.omsi_file.format.omsi_format_dependencydata attribute), 135
dependency_selection (omsi.dataformat.omsi_file.format.omsi_format_dependencydata attribute), 135
dependency_types (omsi.datastructures.dependency_data.dependency_dict attribute), 135
description_value_attribute (omsi.dataformat.omsi_file.format.omsi_format_metadata_collection attribute), 136
dimension_index (omsi.analysis.multivariate_stats.omsi_cx attribute), 118
dimension_index (omsi.analysis.omsi_cx attribute), 92
display() (omsi.analysis.findpeaks.third_party.findpeaks.findpeaks method), 113
driver_base (class in omsi.workflow.driver.base), 209

data_groupname (omsi.dataformat.omsi_file.format.omsi_format_data attribute), 134
data_ready() (omsi.datastructures.analysis_data.parameter_data method), 178
data_set() (omsi.datastructures.analysis_data.parameter_data method), 178
default_db_server_url (omsi.shared.omsi_web_helper.WebHelper attribute), 197
DB_SERVER_URL (omsi.example_web_helper.WebHelper attribute), 197
dependency_dict (class in omsi.datastructures.dependency_data), 180
description_value_attribute (omsi.dataformat.omsi_file.format.omsi_format_metadata_collection attribute), 136
dimension_index (omsi.analysis.multivariate_stats.omsi_cx attribute), 118
dimension_index (omsi.analysis.omsi_cx attribute), 92
display() (omsi.analysis.findpeaks.third_party.findpeaks.findpeaks method), 113
driver_base (class in omsi.workflow.driver.base), 209
email_error_recipients (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
email_success_recipients (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
enable_memory_profiling() (omsi.analysis.analysis_base method), 82
enable_memory_profiling() (omsi.analysis.base.analysis_base method), 100
enable_memory_profiling() (omsi.workflow.common.analysis_task_list method), 207
enable_profile_memory() (omsi.datastructures.run_info_data.run_info_dict method), 182
enable_profile_time_and_usage() (omsi.datastructures.run_info_data.run_info_dict method), 182
enable_time_and_usage_profiling() (omsi.analysis.analysis_base method), 82
enable_time_and_usage_profiling() (omsi.analysis.base.analysis_base method), 100
enable_time_and_usage_profiling() (omsi.workflow.common.analysis_task_list method), 207
error() (omsi.shared.log.log_helper class method), 200
error_handling (omsi.tools.convertToOMSI.ConvertSettings attribute), 220
evaluate_transform_parameter() (in module omsi.shared.data_selection), 188
execute() (omsi.shared.log.log_helper class method), 200
execute() (omsi.analysis.analysis_base method), 82
execute() (omsi.analysis.analysis_generic method), 89
execute() (omsi.analysis.base.analysis_base method), 100
execute() (omsi.analysis.generic.analysis_generic method), 107
execute() (omsi.workflow.driver.base.driver_base method), 209
execute() (omsi.workflow.executor.base.workflow_executor_base method), 216
execute_all() (omsi.analysis.analysis_base class method), 82
execute_all() (omsi.analysis.base.analysis_base class method), 100
execute_analysis() (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 114
execute_analysis() (omsi.analysis.analysis_base method), 83
execute_analysis() (omsi.analysis.analysis_generic method), 89
execute_analysis() (omsi.analysis.base.analysis_base method), 100
execute_analysis() (omsi.analysis.compound_stats.omsi_score_midas.omsi_score_midas method), 130
execute_analysis() (omsi.analysis.compound_stats.omsi_score_pactolus.omsi_score_pactolus method), 131
execute_analysis() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
execute_analysis() (omsi.analysis.findpeaks.experimental.omsi_peakkube.omsi_peakkube method), 114
execute_analysis() (omsi.analysis.findpeaks.omsi_findpeaks_global method), 111
execute_analysis() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global method), 112
execute_analysis() (omsi.analysis.findpeaks.omsi_findpeaks_local method), 111
execute_analysis() (omsi.analysis.findpeaks.omsi_findpeaks_local.omsi_findpeaks_local method), 112
execute_analysis() (omsi.analysis.generic.analysis_generic method), 107
execute_analysis() (omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask.omsi_filter_by_mask method), 126
execute_analysis() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster method), 128
execute_analysis() (omsi.analysis.msi_filtering.omsi_tic_norm method), 121
execute_analysis() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm method), 124
execute_analysis() (omsi.analysis.multivariate_stats.omsi_cx method), 118
execute_analysis() (omsi.analysis.multivariate_stats.omsi_kmeans method), 120
execute_analysis() (omsi.analysis.multivariate_stats.omsi_nmf method), 117
execute_analysis() (omsi.analysis.multivariate_stats.omsi_nmf.omsi_nmf method), 120
execute_analysis() (omsi.analysis.omsi_cx method), 92
execute_analysis() (omsi.analysis.omsi_findpeaks_global method), 90
execute_analysis() (omsi.analysis.omsi_findpeaks_local method), 91
execute_analysis() (omsi.analysis.omsi_kmeans method), 94
execute_analysis() (omsi.analysis.omsi_nmf method), 91
execute_analysis() (omsi.analysis.omsi_tic_norm method), 94
execute_analysis() (omsi.templates.analysis_template.analysis_template method), 225
execute_fpg (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 220
execute_fpl (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 220
execute_nmf (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 221
execute_recursive() (omsi.analysis.analysis_base method), 83
execute_recursive() (omsi.analysis.base.analysis_base method), 101
execute_ticnorm (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 221
exp_groupname (omsi.dataformat.omsi_file.format.omsi_format_experiment attribute), 135
exp_identifier_name (omsi.dataformat.omsi_file.format.omsi_format_experiment attribute), 135

F

file_reader_base (class in omsi.dataformat.file_reader_base), 165
file_reader_base_multidata (class in omsi.dataformat.file_reader_base), 167
file_reader_base_with_regions (class in omsi.dataformat.file_reader_base), 168
file_user (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 221
Find() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
findpeaks (class in omsi.analysis.findpeaks.third_party.findpeaks), 113
flush() (omsi.dataformat.omsi_file.main_file.omsi_file method), 141
format_name (omsi.dataformat.omsi_file.format.omsi_format_msidata attribute), 136
format_name() (omsi.dataformat.file_reader_base.file_reader_base class method), 166
format_option (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 221
format_types (omsi.dataformat.omsi_file.format.omsi_format_msidata attribute), 136
from_function() (omsi.analysis.analysis_generic class method), 89
from_function() (omsi.analysis.generic.analysis_generic class method), 107
from_script_files() (omsi.workflow.common.analysis_task_list class method), 207
from_script_files() (omsi.workflow.executor.base.workflow_executor_base class method), 216
from_scripts() (omsi.workflow.common.analysis_task_list class method), 207
from_scripts() (omsi.workflow.executor.base.workflow_executor_base class method), 216

G

gather() (in module omsi.shared.mpi_helper), 203
gather() (omsi.datastructures.run_info_data.run_info_dict method), 182
generate_thumbnail (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 221
generate_xdmf (omsi.tools.convertToOMSI.ConverteConvertSettings attribute), 221
generateBaseTestFile() (in module omsi.examples.test_multiprocess_slice), 223
generateBaseTestFile() (in module omsi.examples.test_par), 224
generateChunkedTestFile() (in module omsi.examples.test_multiprocess_slice), 223
generateChunkedTestFile() (in module omsi.examples.test_par), 224
generateScript() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
generateScript() (in module omsi.analysis.findpeaks.experimental.pfrun), 114
generateTestFile() (in module omsi.examples.testHDF5Optimization), 223
generateTestFile() (in module omsi.examples.testHDF5Optimization_alignedWrite), 223
get_additional_analysis_dependencies() (omsi.workflow.common.analysis_task_list method), 207
get_all_analysis_data() (omsi.analysis.analysis_base method), 83
get_all_analysis_data() (omsi.analysis.base.analysis_base method), 101
get_all_analysis_data() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 156
get_all_analysis_data() (omsi.workflow.common.analysis_task_list method), 207
get_all_analysis_identifiers() (omsi.workflow.common.analysis_task_list method), 207
get_all_dependency_data() (omsi.analysis.analysis_base method), 83
get_all_dependency_data() (omsi.analysis.base.analysis_base method), 101
get_all_dependency_data() (omsi.dataformat.omsi_file.dependencies.omsi_dependencies_manager method), 158
get_all_dependency_data() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencies method), 162
get_all_dependency_data() (omsi.datastructures.analysis_data.parameter_manager method), 179
get_all_dependency_data() (omsi.workflow.common.analysis_task_list method), 207
get_all_dependency_data_graph() (omsi.dataformat.omsi_file.dependencies.omsi_dependencies_manager method), 159
get_all_dependency_data_graph() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencies method), 162
get_all_dependency_data_recursive() (omsi.dataformat.omsi_file.dependencies.omsi_dependencies_manager method), 160
get_all_dependency_data_recursive() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencies method), 163
get_all_parameter_data() (omsi.analysis.analysis_base method), 83
get_all_parameter_data() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 156
get_all_parameter_data() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_all_run_info() (omsi.analysis.analysis_base method), 83
get_all_run_info() (omsi.analysis.base.analysis_base method), 101
get_all_run_info() (omsi.workflow.executor.base.workflow_executor_base method), 208
get_all_runinfo_data() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 156
get_analyses() (omsi.workflow.executor.base.workflow_executor_base method), 217
get_analysis() (omsi.dataformat.omsi_file.analysis.omsi_analysis_manager method), 156
get_analysis() (omsi.workflow.executor.base.workflow_executor_base method), 217
get_analysis_by_identifier() (omsi.dataformat.omsi_file.analysis.omsi_analysis_manager method), 156
get_analysis_class_from_cl() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
get_analysis_data() (omsi.analysis.analysis_base method), 83
get_analysis_data() (omsi.analysis.base.analysis_base method), 101
get_analysis_data_by_name() (omsi.analysis.analysis_base method), 83
get_analysis_data_by_name() (omsi.analysis.base.analysis_base method), 101
get_analysis_data_shapes_and_types() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
get_analysis_identifier() (omsi.analysis.analysis_base method), 84
get_analysis_identifier() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
get_analysis_identifier() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
get_analysis_identifiers() (omsi.dataformat.omsi_file.analysis.omsi_analysis_manager method), 156
get_analysis_index() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
get_analysis_instance() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
get_analysis_class() (omsi.analysis.analysis_base method), 84
get_analysis_class() (omsi.analysis.base.analysis_base method), 101
get_analysis_type() (omsi.analysis.analysis_base method), 84
get_analysis_type() (omsi.analysis.analysis_generic class method), 89
get_analysis_type() (omsi.analysis.base.analysis_class method), 101
get_analysis_type() (omsi.analysis.generic.analysis_generic class method), 107
get_analysis_type() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
get_axes() (omsi.analysis.analysis_views.analysis_views class method), 109
get_comm_world() (in module omsi.shared.mpi_helper), 203
get_data() (omsi.datastructures.dependency_data.dependency_dict method), 181
get_data_or_default() (omsi.datastructures.analysis_data.parameter_data method), 178
get_dataset_dependencies() (omsi.dataformat.bruckerflex_file.bruckerflex_file method), 174
get_dataset_dependencies() (omsi.dataformat.file_reader_base.file_reader_base_multidata method), 167
get_dataset_dependencies() (omsi.dataformat.file_reader_base.file_reader_base_with_regions method), 168
get_dataset_dependencies() (omsi.dataformat.mzml_file.mzml_file method), 171
get_dataset_metadata() (omsi.dataformat.file_reader_base.file_reader_base method), 166
get_dataset_name() (omsi.dataformat.mzml_file.mzml_file method), 171
get_default_dtypes() (omsi.analysis.analysis_base static method), 84
get_default_dtypes() (omsi.analysis.base.analysis_base static method), 102
get_default_executor() (omsi.workflow.executor.base.workflow_executor_base class method), 217
get_default_executor_class() (omsi.workflow.executor.base.workflow_executor_base class method), 217
get_default_format() (omsi.shared.log.log_helper class method), 201
get_default_metadata_collection() (omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection_manager method), 144
get_default_parameter_groups() (omsi.analysis.analysis_base static method), 84
get_default_parameter_groups() (omsi.analysis.base.analysis_base static method), 102
get_dependency_objecttype() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 164
get_dependency_omsiobject() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencies method), 163
get_dependency_type() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 164
get_dtypes() (omsi.datastructures.analysis_data.data_dtypes static method), 177
get_experiment() (omsi.dataformat.omsi_file.experiment.omsi_experiment_manager method), 142
get_experiment_by_identifier() (omsi.dataformat.omsi_file.experiment.omsi_experiment_manager method), 142
get_experiment_identifier() (omsi.dataformat.omsi_file.experiment.omsi_file_experiment method), 143
get_experiment_path() (omsi.dataformat.omsi_file.experiment.omsi_experiment_manager static method), 142
get_filename() (omsi.dataformat.omsi_file.main_file.omsi_file method), 141
get_files_from_dir() (omsi.dataformat.img_file.img_file class method), 169
get_files_from_dir() (omsi.dataformat.mzml_file.mzml_file method), 171
get_group_description() (omsi.datastructures.analysis_data.parameter_data method), 178
get_group_name() (omsi.datastructures.analysis_data.parameter_data method), 179
get_h5py_datasets() (omsi.dataformat.omsi_file.msidata.omsi_file_msidata method), 151
get_h5py_file() (omsi.dataformat.omsi_file.main_file.omsi_file method), 141
get_h5py_mzdata() (omsi.dataformat.omsi_file.msidata.omsi_file_msidata method), 151
get_h5py_object() (omsi.dataformat.omsi_file.common.omsi_file_common method), 138
get_help_string() (omsi.analysis.analysis_base method), 84
get_help_string() (omsi.analysis.base.analysis_base method), 102
get_instrument_info() (omsi.dataformat.omsi_file.instrument.omsi_file_instrument method), 145
get_instrument_name() (omsi.dataformat.omsi_file.instrument.omsi_file_instrument method), 145
get_metadata() (omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection_manager method), 144
get_metadata_descriptions() (omsi.datastructures.metadata.metadata_data.metadata_dict method), 184
get_metadata_units() (omsi.datastructures.metadata.metadata_data.metadata_dict method), 184
getManagedGroup() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 165
get_memory_profile_info() (omsi.analysis.analysis_base method), 84
get_memory_profile_info() (omsi.analysis.base.analysis_base method), 102
get_metadata() (omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection_manager method), 144
get_metadata_collections() (omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection_manager method), 144
get_metadata_descriptions() (omsi.datastructures.metadata.metadata_data.metadata_dict method), 184
get_metadata_units() (omsi.datastructures.metadata.metadata_data.metadata_dict method), 184

238
get_metadata_values() (omsi.datastructures.metadata.metadata_data.metadata_dict method), 184
get_method_info() (omsi.dataformat.omsi_file.experiment.omsi_file_experiment method), 143
get_method_info() (omsi.dataformat.omsi_file.methods.omsi_methods_manager method), 148
get_method_name() (omsi.dataformat.omsi_file.methods.omsi_file_methods method), 147
get_msidata() (omsi.dataformat.omsi_file.msidata.omsi_msidata_manager method), 154
get_msidata_by_name() (omsi.dataformat.omsi_file.msidata.omsi_msidata_manager method), 154
get_num_analysis() (omsi.dataformat.omsi_file.analysis.omsi_analysis_manager method), 156
get_num_analysis_data() (omsi.analysis.analysis_base.analysis_base method), 102
get_num_analysis_data() (omsi.analysis.analysis_base.analysis_base method), 102
get_num_dependency_data() (omsi.analysis.analysis_base.method), 102
get_num_dependency_data() (omsi.analysis.analysis_base.method), 102
get_num_experiments() (omsi.dataformat.omsi_file.experiment.omsi_experiment_manager method), 142
get_num_items() (omsi.dataformat.omsi_file.common.omsi_file_common class method), 138
get_num_msidata() (omsi.dataformat.omsi_file.msidata.omsi_msidata_manager method), 154
get_num_parameter_data() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_num_parameter_data() (omsi.datastructure_analysis_data.parameter_manager method), 180
get_number_of_datasets() (omsi.dataformat.file_reader_base.file_reader_base method), 166
get_number_of_datasets() (omsi.dataformat.file_reader_base.file_reader_base_multidata method), 168
get_number_of_datasets() (omsi.dataformat.mzml_file.mzml_file method), 171
get_number_of_regions() (omsi.dataformat.file_reader_base.file_reader_base method), 166
get_number_of_regions() (omsi.dataformat.file_reader_base.file_reader_base_with_regions method), 169
get_omsi_analysis_storage() (omsi.analysis.analysis_base.method), 84
get_omsi_analysis_storage() (omsi.analysis.analysis_base.method), 84
get_omsi_dependency() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 165
get_omsi_file_dependencydata() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 164
get_omsi_object() (omsi.dataformat.omsi_file.common.omsi_file_common class method), 138
get_parameter_data() (omsi.analysis.analysis_base method), 84
get_parameter_data() (omsi.analysis.analysis_base method), 84
get_parameter_data() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_parameter_data() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_parameter_data_by_name() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_parameter_data_by_name() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_parameter_help() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 165
get_parameter_name() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 165
get_parameter_names() (omsi.analysis.analysis_base method), 85
get_parameter_names() (omsi.analysis.analysis_base method), 85
get_parameter_names() (omsi.datastructures.analysis_data.parameter_manager method), 180
get_profile_memory() (omsi.datastructures.run_info_data.run_info_dict method), 183
get_profile_stats_object() (omsi.analysis.analysis_base method), 85
get_profile_stats_object() (omsi.analysis.base.analysis_base method), 102
get_profile_stats_object() (omsi.datastructures.run_info_data.run_info_dict method), 183
get_profile_time_and_usage() (omsi.datastructures.run_info_data.run_info_dict method), 183
get_qsliceviewer_options() (omsi.analysis.analysis_views.analysis_views class method), 109
get_qspectrum_viewer_options() (omsi.analysis.analysis_views.analysis_views_class method), 109
get_rank() (in module omsi.shared.mpi_helper), 203
get_real_analysis_type() (omsi.analysis.generic.analysis_generic method), 89
get_real_analysis_type() (omsi.analysis.generic.analysis_generic method), 107
get_region_selection() (omsi.dataformat.file_reader_base.file_reader_base_with_regions method), 169

Index 239
get_regions() (omsi.dataformat.file_reader_base.file_reader_base_with_regions method), 169
get_selection_string() (omsi.dataformat.omsi_file.dependencies.omsi_file_dependencydata method), 165
get_size() (in module omsi.shared.mpi_helper), 203
get_slice() (omsi.analysis.analysis_views.analysis_views class method), 109
get_spectra() (omsi.analysis.analysis_views.analysis_views class method), 109
get_timestamp() (omsi.dataformat.omsi_file.common.omsi_file_common method), 139
get_version() (omsi.dataformat.omsi_file.common.omsi_file_common method), 139
getClustersInfo() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
getCoordIdxB() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
getCoordInfoB() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
getCoordPeaksB() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
global_Mz() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube method), 114
gJobOutput() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
getNearestPeakIndex() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
getnpimage() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
getnppec() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
getPeakCube() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube method), 114
getPFCmd() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
getPixelMap() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
global_log_level (omsi.shared.log.log_helper attribute), 201
greedy_executor (class in omsi.workflow.executor.greedy_executor), 217
groups (omsi.templates.analysis_template.analysis_template attribute), 225

has_default_metadata_collection() (omsi.dataformat.omsi_file.metadata_collection.omsi_metadata_collection_manager method), 145
has_dependencies() (omsi.dataformat.omsi_file.dependencies.omsi_dependencies_manager method), 160
has_instrument_info() (omsi.dataformat.omsi_file.instrument.omsi_instrument_manager method), 146
has_instrument_name() (omsi.dataformat.omsi_file.methods.omsi_file_methods method), 147
has_metadata_collections() (omsi.dataformat.omsi_file.metadata_collection.omsi_metadata_collection_manager method), 145
has_method_info() (omsi.dataformat.omsi_file.methods.omsi_methods_manager method), 148
has_method_name() (omsi.dataformat.omsi_file.methods.omsi_file_methods method), 147
has_omsi_analysis_storage() (omsi.analysis.analysis_base method), 85
has_omsi_analysis_storage() (omsi.analysis.base.analysis_base method), 103
hilbert_curve() (in module omsi.shared.spectrum_layout), 198

img_file (class in omsi.dataformat.img_file), 169
imports_mpi() (in module omsi.shared.mpi_helper), 203
info() (omsi.shared.log.log_helper class method), 201
initialize_argument_parser() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
initialize_argument_parser() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 214
initialized (omsi.shared.log.log_helper attribute), 201
insert() (omsi.workflow.common.analysis_task_list method), 208
instrument_groupname (omsi.dataformat.omsi_file.format.omsi_format_instrument attribute), 135
instrument_mz_name (omsi.dataformat.omsi_file.format.omsi_format_instrument attribute), 135
instrument_name (omsi.dataformat.omsi_file.format.omsi_format_instrument attribute), 135
inv_xy_index_name (omsi.dataformat.omsi_file.format.omsi_format_msidata_partial_cube attribute), 137
io_block_size_limit (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
io_option (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
is_dependency() (omsi.datastructures.analysis_data.parameter_data method), 179
is_managed() (omsi.dataformat.omsi_file.common.omsi_file_common class method), 139
is_mpi_available() (in module omsi.shared.mpi_helper), 203
is_transform_or_reduce() (in module omsi.shared.data_selection), 188
is_valid_dataset() (omsi.dataformat.bruckerflex_file.bruckerflex_file class method), 174
is_valid_dataset() (omsi.dataformat.file_reader_base.file_reader_base class method), 166
is_valid_dataset() (omsi.dataformat.img_file.img_file class method), 169
is_valid_dataset() (omsi.dataformat.mzml_file.mzml_file class method), 171
is_valid_dataset() (omsi.dataformat.omsi_file.main_file.omsi_file_class method), 141
items() (omsi.dataformat.omsi_file.common.omsi_file_common method), 139

J

job_id (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
json_to_transform_reduce_description() (in module omsi.shared.data_selection), 189

K

keys() (omsi.analysis.analysis_base method), 85
keys() (omsi.analysis.base.analysis_base method), 103
keys() (omsi.dataformat.omsi_file.metadata_collection.omsi_file_metadata_collection method), 144
keys() (omsi.datastructures.analysis_data.parameter_manager method), 180

L

locate_analysis() (omsi.analysis.analysis_base_class method), 85
locate_analysis() (omsi.analysis.base.analysis_base_class method), 103
log() (omsi.shared.log.log_helper class method), 201
log_helper (class in omsi.shared.log), 199
log_level_arg_name (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver attribute), 211
log_level_arg_name (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver attribute), 214
log_levels (omsi.shared.log.log_helper attribute), 202
log_var() (omsi.shared.log.log_helper class method), 202

M

main() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
main() (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 114
main() (in module omsi.analysis.findpeaks.experimental.omsi_npg), 116
main() (in module omsi.analysis.findpeaks.experimental.omsi_peakcube), 114
main() (in module omsi.analysis.findpeaks.experimental.pfrun), 114
main() (in module omsi.examples.simple_viewer), 223
main() (in module omsi.examples.test_multiprocess_slice), 223
main() (in module omsi.examples.test_par), 224
main() (in module omsi.examples.test_peakcube), 224
main() (in module omsi.examples.test hdf5_file_read), 224
main() (in module omsi.examples.test HDF5Optimization), 223
main() (in module omsi.examples.test HDF5Optimization_alignedWrite), 223
main() (in module omsi.tools.convertToOMSI), 221
main() (in module omsi.tools.misc.create_peak_cube_ overview), 222
main() (in module omsi.tools.misc.make_thumb), 222
main() (omsi.workflow.driver.base.analysis_driver_base method), 209
main() (omsi.workflow.driver.base.driver_base method), 209

Index 241
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

main() (omsi.workflow.driver.base.workflow_driver_base method), 209
main() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
main() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 214
main() (omsi.workflow.executor.base.workflow_executor_base method), 217
main() (omsi.workflow.executor.greedy_executor.greedy_executor method), 217
make_analysis_identifiers_unique() (omsi.workflow.common.analysis_task_list method), 208
MakeSet() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
metadata (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
metadata_collection_groupname_default (omsi.dataformat.omsi_file.format.omsi_format_metadata_collection attribute), 136
metadata_dict (class in omsi.datastructures.metadata.metadata_data), 184
metadata_ontologies (class in omsi.datastructures.metadata.metadata_ontologies), 185
metadata_value (class in omsi.datastructures.metadata.metadata_data), 184
methods_groupname (omsi.dataformat.omsi_file.format.omsi_format_methods attribute), 136
methods_name (omsi.dataformat.omsi_file.format.omsi_format_methods attribute), 136
methods_old_groupname (omsi.dataformat.omsi_file.format.omsi_format_methods attribute), 136
monitorJob() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
MPI_MESSAGE_TAGS (omsi.shared.mpi_helper.parallel_over_axes attribute), 204
mpi_type_from_dtype() (in module omsi.shared.mpi_helper), 203
myHC() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
MyViewer (class in omsi.examples.simple_viewer), 222
mz_index_name (omsi.dataformat.omsi_file.format.omsi_format_msidata_partial_spectra attribute), 137
mzdata_name (omsi.dataformat.omsi_file.format.omsi_format_msidata attribute), 136
mzml_file (class in omsi.dataformat.mzml_file), 170

N

Name (omsi.analysis.findpeaks.third_party.findpeaks.findpeaks attribute), 113
ndarray() (omsi.datastructures.analysis_data.data_dtypes static method), 177
nlssubprob() (in module omsi.analysis.multivariate_stats.third_party.nmf), 121
nmf() (in module omsi.analysis.multivariate_stats.third_party.nmf), 121
nmf_num_component (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
nmf_num_iter (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
nmf_timeout (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
nmf_use_raw_data (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
Node (class in omsi.analysis.findpeaks.experimental.omsi_npg), 116

O

omsi.analysis (module), 78
omsi.analysis.analysis_views (module), 108
omsi.analysis.base (module), 96
omsi.analysis.compound_stats (module), 130
omsi.analysis.compound_stats.experimental (module), 131
omsi.analysis.compound_stats.omsi_score_midas (module), 130
omsi.analysis.compound_stats.omsi_score_pactolus (module), 131
omsi.analysis.compound_stats.third_party (module), 131
omsi.analysis.findpeaks (module), 110
omsi.analysis.findpeaks.experimental (module), 113
omsi.analysis.findpeaks.experimental.mypeakfinder (module), 117
omsi.analysis.findpeaks.experimental.omsi_lpf (module), 114
omsi.analysis.findpeaks.experimental.omsi_npg (module), 116
omsi.analysis.findpeaks.experimental.omsi_peakcube (module), 114
omsi.analysis.findpeaks.experimental.pfrun (module), 114
omsi.analysis.findpeaks.omsi_findpeaks_global (module), 112
omsi.analysis.findpeaks.omsi_findpeaks_local (module), 112
omsi.analysis.findpeaks.third_party (module), 113
omsi.analysis.findpeaks.third_party.findpeaks (module), 113
omsi.analysis.generic (module), 106
omsi.analysis.msi_filtering (module), 121
omsi.analysis.msi_filtering.experimental (module), 126
omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask (module), 126
omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster (module), 128
omsi.analysis.msi_filtering.omsi_tic_norm (module), 124
omsi.analysis.msi_filtering.third_party (module), 126
omsi.analysis.multivariate_stats (module), 117
omsi.analysis.multivariate_stats.experimental (module), 121
omsi.analysis.multivariate_stats.omsi_nmf (module), 120
omsi.analysis.multivariate_stats.third_party (module), 120
omsi.analysis.multivariate_stats.third_party.nmf (module), 121
omsi.dataformat.bruckerflex_file (module), 172
omsi.dataformat.file_reader_base (module), 165
omsi.dataformat.img_file (module), 169
omsi.dataformat.mzml_file (module), 170
omsi.dataformat.omsi_file (module), 133
omsi.dataformat.omsi_file.analysis (module), 154
omsi.dataformat.omsi_file.common (module), 137
omsi.dataformat.omsi_file.dependencies (module), 158
omsi.dataformat.omsi_file.experiment (module), 141
omsi.dataformat.omsi_file.format (module), 133
omsi.dataformat.omsi_file.instrument (module), 145
omsi.dataformat.omsi_file.main_file (module), 140
omsi.dataformat.omsi_file.metadata_collection (module), 143
omsi.dataformat.omsi_file.methods (module), 147
omsi.dataformat.omsi_file.msidata (module), 148
omsi.datastructures (module), 176
omsi.datastructures.analysis_data (module), 176
omsi.datastructures.dependency_data (module), 180
omsi.datastructures.metadata (module), 184
omsi.datastructures.metadata.metadata_data (module), 184
omsi.datastructures.metadata.metadata_ontologies (module), 185
omsi.datastructures.run_info_data (module), 182
omsi.examples.simple_viewer (module), 222
omsi.examples.test_multiprocess_slice (module), 223
omsi.examples.test_par (module), 224
omsi.examples.test_peakcube (module), 224
omsi.examples.testhdf5_file_read (module), 224
omsi.examples.testHDF5Optimization (module), 223
omsi.examples.testHDF5Optimization_alignedWrite (module), 223
omsi.shared.data_selection (module), 186
omsi.shared.log (module), 199
omsi.shared.mpi_helper (module), 203
omsi.shared.omsi_web_helper (module), 196
omsi.shared.spectrum_layout (module), 198
omsi.templates (module), 224
omsi.templates.analysis_template (module), 224
omsi.tools.convertToOMSI (module), 218
omsi.tools.experimental (module), 222
omsi.tools.misc (module), 222
omsi.tools.misc.create_peak_cube_overview (module), 222
omsi.tools.misc.make_thumb (module), 222
omsi.tools.run_analysis (module), 222
omsi.workflow (module), 206
omsi.workflow.common (module), 206
omsi.workflow.driver (module), 208
omsi.workflow.driver.base (module), 209
omsi.workflow.driver.cl_analysis_driver (module), 209
omsi.workflow.driver.cl_workflow_driver (module), 212
omsi.workflow.executor (module), 215
omsi.workflow.executor.base (module), 215
omsi.workflow.executor.greedy_executor (module), 217
omsi_analysis_manager (class in omsi.dataformat.omsi_file.analysis), 154
omsi Exiting (class in omsi.analysis), 92
omsi_dependencies_manager (class in omsi.dataformat.omsi_file.dependencies), 158
omsi_experiment_manager (class in omsi.dataformat.omsi_file.experiment), 141
omsi_file (class in omsi.dataformat.omsi_file.main_file), 140
omsi_file_analysis (class in omsi.dataformat.omsi_file.analysis), 156
omsi_file_common (class in omsi.dataformat.omsi_file.common), 137
omsi_file_dependencies (class in omsi.dataformat.omsi_file.dependencies), 160
omsi_file_dependencydata (class in omsi.dataformat.omsi_file.dependencies), 164
omsi_file_experiment (class in omsi.dataformat.omsi_file.experiment), 142
omsi_file_instrument (class in omsi.dataformat.omsi_file.instrument), 145
omsi_file_metadata_collection (class in omsi.dataformat.omsi_file.metadata_collection), 143
omsi_file_methods (class in omsi.dataformat.omsi_file.methods), 147
omsi_file_msidata (class in omsi.dataformat.omsi_file.msidata), 148
omsi_file_object_manager (class in omsi.dataformat.omsi_file.common), 140
omsi_filter_by_mask (class in omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask), 126
omsi_findpeaks_global (class in omsi.analysis), 90
omsi_findpeaks_global (class in omsi.analysis.findpeaks), 111
omsi_findpeaks_global (class in omsi.analysis.findpeaks.omsi_findpeaks_global), 112
omsi_findpeaks_local (class in omsi.analysis), 90
omsi_findpeaks_local (class in omsi.analysis.findpeaks), 111
omsi_findpeaks_local (class in omsi.analysis.findpeaks.omsi_findpeaks_local), 112
omsi_format_analysis (class in omsi.dataformat.omsi_file.format), 133
omsi_format_common (class in omsi.dataformat.omsi_file.format), 133
omsi_format_data (class in omsi.dataformat.omsi_file.format), 134
omsi_format_dependencies (class in omsi.dataformat.omsi_file.format), 134
omsi_format_dependencydata (class in omsi.dataformat.omsi_file.format), 134
omsi_format_experiment (class in omsi.dataformat.omsi_file.format), 135
omsi_format_file (class in omsi.dataformat.omsi_file.format), 135
omsi_format_instrument (class in omsi.dataformat.omsi_file.format), 135
omsi_format_metadata_collection (class in omsi.dataformat.omsi_file.format), 135
omsi_format_methods (class in omsi.dataformat.omsi_file.format), 136
omsi_format_msidata (class in omsi.dataformat.omsi_file.format), 136
omsi_format_msidata_partial_cube (class in omsi.dataformat.omsi_file.format), 136
omsi_format_msidata_partial_spectra (class in omsi.dataformat.omsi_file.format), 137
omsi_instrument_manager (class in omsi.dataformat.omsi_file.instrument), 146
omsi_kmeans (class in omsi.analysis), 94
omsi_kmeans (class in omsi.analysis.multivariate_stats), 120
omsi_lpf (class in omsi.analysis.findpeaks.experimental.omsi_lpf), 114
omsi_mask_by_cluster (class in omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster), 128
omsi_metadata_collection_manager (class in omsi.dataformat.omsi_file.metadata_collection), 144
omsi_methods_manager (class in omsi.dataformat.omsi_file.methods), 147
omsi_msidata_manager (class in omsi.dataformat.omsi_file.msidata), 151
omsi_nmf (class in omsi.analysis), 91
omsi_nmf (class in omsi.analysis.multivariate_stats), 117
omsi_nmf (class in omsi.analysis.multivariate_stats.omsi_nmf), 120
omsi_npg (class in omsi.analysis.findpeaks.experimental.omsi_npg), 116
omsi_output_file (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
omsi_peackcube (class in omsi.analysis.findpeaks.experimental.omsi_peackcube), 114
omsi_score_midas (class in omsi.analysis.compound_stats.omsi_score_midas), 130
omsi_score_pactolus (class in omsi.analysis.compound_stats.omsi_score_pactolus), 131
omsi_tic_norm (class in omsi.analysis), 94
omsi_tic_norm (class in omsi.analysis.msi_filtering), 121
omsi_tic_norm (class in omsi.analysis.msi_filtering.omsi_tic_norm), 124
ontology_value_attribute (omsi.dataformat.omsi_file.format.omsi_format_metadata_collection attribute), 136
output_save_arg_name (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver attribute), 211
output_save_arg_name (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver attribute), 214

P
parallel_over_axes (class in omsi.shared.mpi_helper), 203
parameter_data (class in omsi.datastructures.analysis_data), 177
parameter_manager (class in omsi.datastructures.analysis_data), 179
parse_cl_arguments() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
parse_cl_arguments() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 214
parse_input_args() (omsi.tools.convertToOMSI.ConvertSettings class method), 221
parse_path_string() (omsi.dataformat.omsi_file.common.omsi_file_common static method), 139
peakdet() (omsi.analysis.findpeaks.third_party.findpeaks.findpeaks method), 113
perform_reduction() (in module omsi.shared.data_selection), 189
plot_2d_spectrum_as_image() (in module omsi.shared.spectrum_layout), 198
print_help() (in module omsi.tools.misc.create_peak_cube_overview), 222
print_help() (omsi.tools.convertToOMSI.ConvertSettings class method), 221
print_memory_profiles() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 215
print_settings() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 211
print_settings() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 215
print_time_and_usage_profiles() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 215
printHelp() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
printHelp() (in module omsi.analysis.findpeaks.experimental.pfrun), 114
printHelp() (in module omsi.examples.test_peakcube), 224
printHelp() (in module omsi.examples.testhdf5_file_read), 224
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

printHelp() (in module omsi.examples.testHDF5Optimization), 223
printHelp() (in module omsi.examples.testHDF5Optimization_alignedWrite), 223
profile_arg_name (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver attribute), 215
profile_mem_arg_name (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver attribute), 215

Q

queuePCjob() (in module omsi.analysis.findpeaks.experimental.pfrun), 114

R

RawDescriptionDefaultHelpArgParseFormatter (class in omsi.workflow.common), 206
read_from_omsi_file() (omsi.analysis.analysis_base method), 85
read_from_omsi_file() (omsi.analysis.analysis_generic method), 90
read_from_omsi_file() (omsi.analysis.base.analysis_base method), 103
read_from_omsi_file() (omsi.analysis.generic.analysis_generic method), 107
record_execute_analysis_outputs() (omsi.analysis.analysis_base method), 86
record_execute_analysis_outputs() (omsi.analysis.base.analysis_base method), 104
record_execute_analysis_outputs() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
record_execute_analysis_outputs() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube method), 114
record_execute_analysis_outputs() (omsi.analysis.msi_filtering.omsi_tic_norm method), 122
record_execute_analysis_outputs() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm method), 124
record_execute_analysis_outputs() (omsi.analysis.msi_tic_norm method), 94
record_postexecute() (omsi.datastructures.run_info_data.run_info_dict method), 183
record_preexecute() (omsi.datastructures.run_info_data.run_info_dict method), 183
recorded_warnings (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
recreate_analysis() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
reduction_allowed_numpy_function (in module omsi.shared.data_selection), 189
region_option (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
register_file_with_db() (omsi.shared.omsi_web_helper.WebHelper static method), 197
reinterpolate_spectrum() (in module omsi.shared.spectrum_layout), 199
remove_output_target() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 212
remove_output_target() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 215
reset_analysis_object() (omsi.workflow.driver.cl_analysis_driver.cl_analysis_driver method), 212
reset_workflow_executor_object() (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver method), 215
restore_analysis() (omsi.dataformat.omsi_file.analysis.omsi_file_analysis method), 157
results_ready() (omsi.analysis.analysis_base method), 86
results_ready() (omsi.analysis.base.analysis_base method), 104
run() (omsi.shared.mpi_helper.parallel_over_axes method), 205
run_dict (class in omsi.datastructures.run_info_data), 182
run_lpf() (in module omsi.analysis.findpeaks.experimental.pfrun), 114
run_npg() (in module omsi.analysis.findpeaks.experimental.pfrun), 114
run_peakcube() (in module omsi.analysis.findpeaks.experimental.pfrun), 114

S

s_mz_from_acqu() (omsi.dataformat.bruckerflex_file.bruckerflex_file static method), 174
s_read_acqu() (omsi.dataformat.bruckerflex_file.bruckerflex_file static method), 174
s_read_fid() (omsi.dataformat.bruckerflex_file.bruckerflex_file static method), 174
s_read_spotlist() (omsi.dataformat.bruckerflex_file.bruckerflex_file static method), 174
s_spot_from_dir() (omsi.dataformat.bruckerflex_file.bruckerflex_file static method), 175
same_file() (omsi.dataformat.omsi_file.common.omsi_file_common static method), 139
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

SCHEDULES (omsi.shared.mpi_helper.parallel_over_axes attribute), 205
script_arg_name (omsi.workflow.driver.cl_workflow_driver.cl_workflow_driver_attribute), 215
selection_string_to_object() (in module omsi.shared.data_selection), 190
selection_to_indexlist() (in module omsi.shared.data_selection), 191
selection_to_string() (in module omsi.shared.data_selection), 191
selection_type (in module omsi.shared.data_selection), 191
send_email() (omsi.shared.omsi_web_helper.WebHelper static method), 197
set_analysis_identifier() (omsi.analysis.analysis_base method), 86
set_analysis_identifier() (omsi.analysis.base.analysis_base method), 104
set_apache_acl() (omsi.shared.omsi_web_helper.WebHelper static method), 198
set_dataset_selection() (omsi.dataformat.file_reader_base.file_reader_base_multidata method), 168
set_dataset_selection() (omsi.dataformat.mzml_file.mzml_file method), 171
set_experiment_identifier() (omsi.dataformat.omsi_file.experiment.omsi_file_experiment method), 143
set_fill_space() (omsi.dataformat.omsi_file.msidata.omsi_file_msidata method), 151
set_fill_spectra() (omsi.dataformat.omsi_file.msidata.omsi_file_msidata method), 151
set_instrument_name() (omsi.dataformat.omsi_file.instrument.omsi_file_instrument method), 146
set_log_level() (omsi.shared.log.log_helper class method), 202
set_method_name() (omsi.dataformat.omsi_file.methods.omsi_file_methods method), 147
set_parameter_default_value() (omsi.datastructures.analysis_data.parameter_manager method), 180
set_parameter_values() (omsi.analysis.analysis_base method), 86
set_parameter_values() (omsi.analysis.base.analysis_base method), 104
set_region_selection() (omsi.dataformat.bruckerflex_file.bruckerflex_file method), 176
set_region_selection() (omsi.dataformat.file_reader_base.file_reader_base_with_regions method), 169
set_undefined_analysis_identifiers() (omsi.workflow.common.analysis_task_list method), 208
setup_logging() (omsi.shared.log.log_helper class method), 202
shape_name (omsi.dataformat.omsi_file.format.omsi_format_msidata_partial_cube attribute), 137
size() (omsi.dataformat.file_reader_base.file_reader_base_class method), 167
size() (omsi.dataformat.img_file.img_file class method), 170
size() (omsi.dataformat.mzml_file.mzml_file class method), 171
sliceSelect() (in module omsi.examples.test_multiprocess_slice), 223
sliceSelect() (in module omsi.examples.test_par), 224
sliding_window_minimum() (omsi.analysis.findpeaks.third_party.findpeaks.findpeaks method), 113
smoothListGaussian() (omsi.analysis.findpeaks.third_party.findpeaks.findpeaks method), 113
spectraSelect() (in module omsi.examples.test_multiprocess_slice), 223
spectraSelect() (in module omsi.examples.test_par), 224
spectrum_iter() (omsi.dataformat.file_reader_base.file_reader_base method), 167
spectrum_iter() (omsi.dataformat.img_file.img_file method), 170
spectrum_iter() (omsi.dataformat.mzml_file.mzml_file method), 171
splitLabelsList() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
stop() (in module omsi.analysis.findpeaks.experimental.mypeakfinder), 117
stop() (in module omsi.analysis.findpeaks.experimental.omsi_peackube), 114
stop() (in module omsi.analysis.findpeaks.experimental.pfrun), 114
str_type (omsi.dataformat.omsi_file.format.omsi_format_common attribute), 133
str_type_unicode (omsi.dataformat.omsi_file.format.omsi_format_common attribute), 133
string_to_structime() (omsi.datastructures.run_info_data.run_info_dict static method), 184
string_to_time() (omsi.datastructures.run_info_data.run_info_dict static method), 184
suggest_chunking() (omsi.tools.convertToOMSI.ConvertFiles static method), 219
suggest_chunkings_for_files() (omsi.tools.convertToOMSI.ConvertFiles static method), 219
suggest_file_chunkings (omsi.tools.convertToOMSI.ConvertSettings attribute), 221

Index
super_users (omsi.shared.omsi_web_helper.WebHelper attribute), 198
supports_multidata() (omsi.dataformat.file_reader_base.file_reader_base class method), 167
supports_multidata() (omsi.dataformat.file_reader_base.file_reader_base_multidata class method), 168
supports_multiexperiment() (omsi.dataformat.file_reader_base.file_reader_base class method), 167
supports_regions() (omsi.dataformat.file_reader_base.file_reader_base class method), 167
supports_regions() (omsi.dataformat.file_reader_base.file_reader_base_with_regions class method), 169
supports_slice() (omsi.analysis.analysis_views.analysis_views.class method), 110
supports_spectra() (omsi.analysis.analysis_views.analysis_views.class method), 110

t

test() (omsi.dataformat.mzml_file.mzml_file class method), 172
test_mpi_available() (in module omsi.shared.mpi_helper), 205
timestamp_attribute (omsi.dataformat.omsi_file.format.omsi_format_common attribute), 134
transform_and_reduce_data() (in module omsi.shared.data_selection), 191
transform_data_single() (in module omsi.shared.data_selection), 192
transform_datachunk() (in module omsi.shared.data_selection), 192
transform_reduce_description_to_json() (in module omsi.shared.data_selection), 193
transformation_allowed_numpy_dual_data (in module omsi.shared.data_selection), 193
transformation_allowed_numpy_single_data (in module omsi.shared.data_selection), 194
transformation_type (in module omsi.shared.data_selection), 195
type_attribute (omsi.dataformat.omsi_file.format.omsi_format_common attribute), 134

U

 Union() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg method), 116
unit_value_attribute (omsi.dataformat.omsi_file.format.omsi_format_metadata_collection attribute), 136
update() (omsi.workflow.common.analysis_task_list method), 208
update_analysis_parameters() (omsi.analysis.analysis_base method), 86
update_analysis_parameters() (omsi.analysis.base.analysis_base method), 104
update_job_status() (omsi.shared.omsi_web_helper.WebHelper static method), 198
user_additional_chunks (omsi.tools.convertToOMSI.ConvertSettings attribute), 221
UserInput (class in omsi.shared.omsi_web_helper), 196
userinput_with_timeout() (omsi.shared.omsi_web_helper.UserInput static method), 196
userinput_with_timeout_defautl() (omsi.shared.omsi_web_helper.UserInput static method), 196
userinput_with_timeout_windows() (omsi.shared.omsi_web_helper.UserInput static method), 197

V

v_qmz (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 115
v_qmz() (omsi.analysis.analysis_base class method), 86
v_qmz() (omsi.analysis.analysis_generic class method), 90
v_qmz() (omsi.analysis.base.analysis_base class method), 104
v_qmz() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg class method), 116
v_qmz() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube class method), 114
v_qmz() (omsi.analysis.findpeaks.omsi_findpeaks_global class method), 111
v_qmz() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global class method), 112
v_qmz() (omsi.analysis.findpeaks.omsi_findpeaks_local class method), 111
v_qmz() (omsi.analysis.findpeaks.omsi_findpeaks_local.omsi_findpeaks_local class method), 112
v_qmz() (omsi.analysis.generic.analysis_generic class method), 107
v_qmz() (omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask.omsi_filter_by_mask class method), 126
v_qmz() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster class method), 128
v_qmz() (omsi.analysis.msi_filtering.omsi_tic_norm class method), 122
v_qmz() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm class method), 124
v_qmz() (omsi.analysis.multivariate_stats.omsi_omsi class method), 118
v_qmz() (omsi.analysis.multivariate_stats.omsi_nmf class method), 117
v_qmz() (omsi.analysis.multivariate_stats.omsi_nmf.omsi_nmf class method), 120
v_qmz() (omsi.analysis.omsi_omsi class method), 92
v_qmz() (omsi.analysis.omsi_findpeaks_global class method), 90
v_qmz() (omsi.analysis.omsi_findpeaks_local class method), 91
v_qmz() (omsi.analysis.omsi_nmf class method), 91
v_qmz() (omsi.analysis.omsi_tic_norm class method), 94
v_qmz() (omsi.templates.analysis_template.analysis_template class method), 225
v_qslice (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 115
v_qslice() (omsi.analysis.analysis_base class method), 87
v_qslice() (omsi.analysis.analysis_generic class method), 90
v_qslice() (omsi.analysis.base.analysis_base class method), 105
v_qslice() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg class method), 116
v_qslice() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube class method), 114
v_qslice() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global class method), 111
v_qslice() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global class method), 112
v_qslice() (omsi.analysis.findpeaks.omsi_findpeaks_local.omsi_findpeaks_local class method), 112
v_qslice() (omsi.analysis.generic.analysis_generic class method), 107
v_qslice() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster class method), 127
v_qslice() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster class method), 128
v_qslice() (omsi.analysis.msi_filtering.omsi_tic_norm class method), 122
v_qslice() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm class method), 124
v_qslice() (omsi.analysis.multivariate_stats.omsi_omsi class method), 118
v_qslice() (omsi.analysis.multivariate_stats.omsi_nmf class method), 117
v_qslice() (omsi.analysis.multivariate_stats.omsi_nmf.omsi_nmf class method), 120
v_qslice() (omsi.analysis.omsi_omsi class method), 92
v_qslice() (omsi.analysis.omsi_findpeaks_global class method), 90
v_qslice() (omsi.analysis.omsi_findpeaks_local class method), 91
v_qslice() (omsi.analysis.omsi_nmf class method), 91
v_qslice() (omsi.analysis.omsi_tic_norm class method), 95
v_qslice() (omsi.templates.analysis_template.analysis_template class method), 225
v_qslice_viewer_options (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 115
v_qslice_viewer_options() (omsi.analysis.analysis_base class method), 87
v_qslice_viewer_options() (omsi.analysis.analysis_generic class method), 90
v_qslice_viewer_options() (omsi.analysis.base.analysis_base class method), 105
v_qslice_viewer_options() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg class method), 116
v_qslice_viewer_options() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube class method), 114
v_qslice_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global class method), 111
v_qslice_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global class method), 112
v_qslice_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_local.omsi_findpeaks_local class method), 111
v_qslice_viewer_options() (omsi.analysis.generic.analysis_generic class method), 107
v_qslice_viewer_options() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster class method), 127
v_qslice_viewer_options() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster class method), 128

Index 249
BASTet: Berkeley Analysis and Storage Toolkit, Release devel

v_qslice_viewer_options() (omsi.analysis.msi_filtering.omsi_tic_norm.class method), 129
v_qslice_viewer_options() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm.class method), 122
v_qslice_viewer_options() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm.class method), 125
v_qslice_viewer_options() (omsi.analysis.multivariate_stats.omsi_cx.class method), 119
v_qslice_viewer_options() (omsi.analysis.multivariate_stats.omsi_nmf.class method), 117
v_qslice_viewer_options() (omsi.analysis.multivariate_stats.omsi_nmf.omsi_nmf.class method), 120
v_qslice_viewer_options() (omsi.analysis.omsi_cx.class method), 93
v_qslice_viewer_options() (omsi.analysis.omsi_findpeaks_global.class method), 90
v_qslice_viewer_options() (omsi.analysis.omsi_findpeaks_local.class method), 91
v_qslice_viewer_options() (omsi.analysis.omsi_nmf.class method), 91
v_qslice_viewer_options() (omsi.analysis.omsi_tic_norm.class method), 95
v_qslice_viewer_options() (omsi.templates.analysis_template.analysis_template.class method), 226
v_qspectrum (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 115
v_qspectrum() (omsi.analysis.analysis_base.class method), 88
v_qspectrum() (omsi.analysis.analysis_generic.class method), 90
v_qspectrum() (omsi.analysis.base.analysis_base.class method), 105
v_qspectrum() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg.class method), 116
v_qspectrum() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube.class method), 114
v_qspectrum() (omsi.analysis.findpeaks.omsi_findpeaks_global.class method), 111
v_qspectrum() (omsi.analysis.findpeaks_global.omsi_findpeaks_global.class method), 112
v_qspectrum() (omsi.analysis.findpeaks_local.omsi_findpeaks_local.class method), 111
v_qspectrum() (omsi.analysis.generic.analysis_generic.class method), 107
v_qspectrum() (omsi.analysis.msi_filtering.experimental.omsi_filter_by_mask.omsi_filter_by_mask.class method), 127
v_qspectrum() (omsi.analysis.msi_filtering.experimental.omsi_mask_by_cluster.omsi_mask_by_cluster.class method), 129
v_qspectrum() (omsi.analysis.msi_filtering.omsi_tic_norm.class method), 123
v_qspectrum() (omsi.analysis.msi_filtering.omsi_tic_norm.omsi_tic_norm.class method), 125
v_qspectrum() (omsi.analysis.multivariate_stats.omsi_cx.class method), 119
v_qspectrum() (omsi.analysis.multivariate_stats.omsi_nmf.class method), 117
v_qspectrum() (omsi.analysis.multivariate_stats.omsi_nmf.omsi_nmf.class method), 120
v_qspectrum() (omsi.analysis.omsi_cx.class method), 93
v_qspectrum() (omsi.analysis.omsi_findpeaks_global.class method), 90
v_qspectrum() (omsi.analysis.omsi_findpeaks_local.class method), 91
v_qspectrum() (omsi.analysis.omsi_nmf.class method), 91
v_qspectrum() (omsi.analysis.omsi_tic_norm.class method), 95
v_qspectrum() (omsi.templates.analysis_template.analysis_template.class method), 226
v_spectrum_viewer_options (in module omsi.analysis.findpeaks.experimental.omsi_lpf), 115
v_spectrum_viewer_options() (omsi.analysis.analysis_base.class method), 88
v_spectrum_viewer_options() (omsi.analysis.analysis_generic.class method), 90
v_spectrum_viewer_options() (omsi.analysis.base.analysis_base.class method), 106
v_spectrum_viewer_options() (omsi.analysis.findpeaks.experimental.omsi_npg.omsi_npg.class method), 117
v_spectrum_viewer_options() (omsi.analysis.findpeaks.experimental.omsi_peakcube.omsi_peakcube.class method), 114
v_spectrum_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_global.class method), 111
v_spectrum_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_global.omsi_findpeaks_global.class method), 112
v_spectrum_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_local.class method), 111
v_spectrum_viewer_options() (omsi.analysis.findpeaks.omsi_findpeaks_local.omsi_findpeaks_local.class method),
Index 251