mh-utils

Release 0.2.2

Utilities for handing ancillary files produced by MassHunter.

Dominic Davis-Foster

Apr 25, 2024

Contents

The current utilities are as follows:

- *mh_utils.cef_parser*: Parse Agilent MassHunter Compound Exchange Format files (`*.cef` files).
- *mh_utils.csv_parser*: Parser for CSV result files produced by MassHunter Qualitative.
- *mh_utils.worklist_parser*: Parse Agilent MassHunter Worklists (`*.wkl` files).

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install mh_utils --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install mh_utils
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/PyMassSpec/mh_utils@master --user
```

Chapter

TWO

mh_utils.utils

General utility functions.

Functions:

as_path(val)	Returns `val` as a `PureWindowsPath`, or `None` if the value is empty/`None`/`False`.
camel_to_snake(name)	Convert `name` from `CamelCase` to `snake_case`.
element_to_bool(val)	Returns the boolean representation of `val`.
make_timedelta(minutes)	Construct a timedelta from a value in minutes.
strip_string(val)	Returns `val` as a string, without any leading or trailing whitespace.

as_path(*val*)

Returns `val` as a `PureWindowsPath`, or `None` if the value is empty/`None`/`False`.

Parameters **val** (`Any`) – The value to convert to a path

Return type `Optional`[`PureWindowsPath`]

camel_to_snake(*name*)

Convert `name` from `CamelCase` to `snake_case`.

Parameters **name** (`str`) – The `CamelCase` string to convert to `snake_case`.

Return type `str`

element_to_bool(*val*)

Returns the boolean representation of `val`.

Values of `-1` are counted as `True` for the purposes of this function.

`True` values are `'y'`, `'yes'`, `'t'`, `'true'`, `'on'`, `'1'`, `1`, `-1`, and `'-1'`.

`False` values are `'n'`, `'no'`, `'f'`, `'false'`, `'off'`, `'0'`, and `0`.

Raises `ValueError` if 'val' is anything else.

Return type `bool`

make_timedelta(*minutes*)

Construct a timedelta from a value in minutes.

Parameters **minutes** (`Union`[`float`, `timedelta`])

Return type `timedelta`

Changed in version 0.1.0: Moved from *mh_utils.cef_parser*.

strip_string(*val*)
Returns `val` as a string, without any leading or trailing whitespace.

Parameters **val** (`str`)

Return type `str`

Chapter

THREE

mh_utils.xml

Functions and classes for handling XML files.

Classes:

XMLFileMixin()	ABC mixin to provide a function for instantiating the class from an XML file.

Functions:

get_validated_tree(xml_file[, schema_file])	Returns a validated lxml objectify from the given XML file, validated against the schema file.

class XMLFileMixin

Bases: ABC

ABC mixin to provide a function for instantiating the class from an XML file.

Methods:

from_xml(element)	Construct an object from an XML element.
from_xml_file(filename)	Generate an instance of this class by parsing an from an XML file.

abstract classmethod from_xml(*element*)

Construct an object from an XML element.

classmethod from_xml_file(*filename*)

Generate an instance of this class by parsing an from an XML file.

Parameters **filename** (Union[str, Path, PathLike]) – The filename of the XML file.

get_validated_tree(*xml_file*, *schema_file=None*)

Returns a validated lxml objectify from the given XML file, validated against the schema file.

Parameters

- **xml_file** (Union[str, Path, PathLike]) – The XML file to validate.
- **schema_file** (Union[str, Path, PathLike, None]) – The schema file to validate against. Default None.

Return type _ElementTree

Returns An lxml ElementTree object. When .getroot() us called on the tree the root will be an instance of lxml.objectify.ObjectifiedElement.

Chapter

FOUR

mh_utils.cef_parser

Parser for MassHunter Compound Exchange Format `.cef` files.

A CEF file represents a file identified in LC-MS data by MassHunter Qualitative. It consists of a list of compounds encapsulated in a *CompoundList*.

A *CompoundList* consists of *Compound* objects representing the individual compounds identified in the data. Each *Compound* object contains information on the location of that compound within the LC data (*location*), the scores indicating the confidence of the match (*compound_scores*), a list of possible matching compounds (*results*), and the matching mass spectrum extracted from the LC-MS data (*spectra*).

The following diagram represents this structure:

- *CompoundList*
  - *Compound*
    - *Compound.algo* ⇒ `str`
    - *Compound.location* ⇒ `Optional` [*LocationDict*]
    - *Compound.compound_scores* ⇒ `Optional` [`Dict` [`str`, *Score*]]
    - *Compound.results* ⇒ `List`
      - *Molecule*
      - Another *Molecule*
      - `...`
    - *Compound.spectra* ⇒ `List`
      - *Spectrum*
      - Another *Spectrum*
      - `...`
  - Another *Compound*
  - `...`

Classes:

Compound([algo, location, compound_scores, ...])	Represents a compound identified in mass spectral data by MassHunter Qualitative.
CompoundList([instrument, compounds])	A list of Compound objects parsed from a CEF file.
Device(device_type, number)	Represents the device that acquired a *Spectrum*.
Flag(string, severity)	Represents a flag in a score, to warn that the identification of a compound is poor.
LocationDict	TypedDict representing the location of a spectrum within mass spectrometry data.
Molecule(name[, formula, matches])	Represents a molecule in a CEF file.
Peak(x, rx, y[, charge, label])	A peak in a Mass Spectrum.
RTRange([start, end])	Represents an <RTRange> element from a CEF file.
Score(score[, flag_string, flag_severity])	A score indicating how well the compound matches the observed spectrum.
Spectrum([spectrum_type, algorithm, ...])	Agilent CEF Spectrum.

Functions:

make_timedelta(minutes)	Construct a timedelta from a value in minutes.
parse_cef(filename)	Construct an *CompoundList* object from the given .cef file.
parse_compound_scores(element)	Parse a <CompoundScores> element into a mapping of algorithms to scores.
parse_match_scores(element)	Parse a <MatchScores> element into a mapping of algorithms to scores.

class Compound(*algo=''*, *location=None*, *compound_scores=None*, *results=None*, *spectra=None*)

Bases: Dictable

Represents a compound identified in mass spectral data by MassHunter Qualitative.

Parameters

- **algo** (str) – The algorithm used to identify the compound. Default ''.
- **location** (Optional[*LocationDict*]) – A dictionary of information to locate the compound in the spectral data. Default None.
- **compound_scores** (Optional[Dict[str, *Score*]]) – A dictionary of compound scores. Default None.
- **results** (Optional[Sequence[*Molecule*]]) – A list of molecules that match the spectrum. Default None.
- **spectra** (Optional[Sequence[*Spectrum*]]) – A list of spectra for the compound. Default None.

Methods:

__repr__()	Returns a string representation of the *Compound*.
__str__()	Returns the *Compound* as a string.
from_xml(element)	Construct a *Compound* object from an XML element.

Attributes:

algo	The algorithm used to identify the compound.
compound_scores	A dictionary of compound scores.
location	A dictionary of information to locate the compound in the spectral data.
results	A list of molecules that match the spectrum.
spectra	A list of spectra for the compound.

__repr__()
Returns a string representation of the *Compound*.

Return type str

__str__()
Returns the *Compound* as a string.

Return type str

algo
Type: str

The algorithm used to identify the compound.

compound_scores
Type: Dict[str, *Score*]

A dictionary of compound scores.

classmethod from_xml(*element*)
Construct a *Compound* object from an XML element.

Parameters **element** (ObjectifiedElement) – a Compound XML element from a CEF file.

Return type *Compound*

location
Type: *LocationDict*

A dictionary of information to locate the compound in the spectral data.

results
Type: List[*Molecule*]

A list of molecules that match the spectrum.

spectra
Type: List[*Spectrum*]

A list of spectra for the compound.

class CompoundList(*instrument=''*, *compounds=None*)

Bases: NamedList

A list of Compound objects parsed from a CEF file.

The full list API is available for this class.

Parameters

- **instrument** (str) – String identifying the instrument that acquired the data. Default ''.
- **compounds** (Optional[Iterable[*Compound*]]) – List of compounds identified in the mass spectrometry data. Default None.

Methods:

__repr__()	Return a string representation of the NamedList.
__str__()	Returns the list as a string.
from_xml(element)	Construct a *CompoundList* object from an XML element.

Attributes:

instrument	The type of instrument that obtained the data, e.g.

__repr__()

Return a string representation of the NamedList.

Return type str

__str__()

Returns the list as a string.

Return type str

classmethod from_xml(*element*)

Construct a *CompoundList* object from an XML element.

Parameters **element** (ObjectifiedElement) – The XML element to parse the data from.

Return type *CompoundList*

instrument

Type: str

The type of instrument that obtained the data, e.g. "LCQTOF".

class Device(*device_type*, *number*)

Bases: object

Represents the device that acquired a *Spectrum*.

Parameters

- **device_type** (str) – String identifying the type of device.
- **number** (int)

Attributes:

device_type	String identifying the type of device.
number	

Methods:

from_dict(d)	Construct an instance of *Device* from a dictionary.
from_xml(element)	Construct a *Device* object from an XML element.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Device* object.

device_type

Type: str

String identifying the type of device.

classmethod from_dict(*d*)

Construct an instance of *Device* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

classmethod from_xml(*element*)

Construct a *Device* object from an XML element.

Parameters **element** (ObjectifiedElement) – a <Device> XML element from a CEF file

Return type *Device*

number

Type: int

to_dict(*convert_values=False*)

Returns a dictionary containing the contents of the *Device* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

class Flag(*string: str*, *severity: int*)

Bases: str

Represents a flag in a score, to warn that the identification of a compound is poor.

Parameters

- **string** – The text of the flag
- **severity** – The severity of the flag

Methods:

__bool__()	Returns a boolean representation of the *Flag*.
__eq__(other)	Return `self == other`.
__ne__(other)	Return `self != other`.

__bool__()

Returns a boolean representation of the *Flag*.

Return type bool

__eq__(*other*)

Return `self == other`.

Return type bool

__ne__(*other*)

Return `self != other`.

Return type bool

typeddict LocationDict

Bases: TypedDict

TypedDict representing the location of a spectrum within mass spectrometry data.

Optional Keys

- **m** (float) – the accurate mass of the compound, determined from the observed mass spectrum.
- **rt** (float) – The retention time at which the compound was detected.
- **a** (float) – The area of the peak in the EIC.
- **y** (float) – The height of the peak in the EIC.

class Molecule(*name*, *formula=None*, *matches=None*)

Bases: Dictable

Represents a molecule in a CEF file.

Parameters

- **name** (str) – The name of the compound
- **formula** (Union[str, Formula, None]) – The formula of the compound. If a string it must be parsable by chemistry_tools.formulae.Formula. Default None.
- **matches** (Optional[Dict[str, *Score*]]) – Dictionary of algo: score match values. Default None.

Methods:

__repr__()	Returns a string representation of the *Molecule*.
__str__()	Returns the molecule as a string.
from_xml(element)	Construct a *Molecule* object from an XML element.

__repr__()

Returns a string representation of the *Molecule*.

Return type str

__str__()

Returns the molecule as a string.

Return type str

classmethod from_xml(*element*)

Construct a *Molecule* object from an XML element.

Parameters **element** (ObjectifiedElement) – a Molecule XML element

Return type *Molecule*

parse_cef(*filename*)

Construct an *CompoundList* object from the given .cef file.

Parameters **filename** (Union[str, Path, PathLike]) – The filename of the CEF file to read.

Return type *CompoundList*

parse_compound_scores(*element*)

Parse a <CompoundScores> element into a mapping of algorithms to scores.

Parameters **element** (ObjectifiedElement) – a CompoundScores XML element.

Return type Dict[str, *Score*]

parse_match_scores(*element*)

Parse a <MatchScores> element into a mapping of algorithms to scores.

Parameters **element** (ObjectifiedElement) – a MatchScores XML element.

Return type Dict[str, *Score*]

class Peak(*x*, *rx*, *y*, *charge=0*, *label=''*)
Bases: object

A peak in a Mass Spectrum.

Parameters

- **x** (float)
- **rx** (float)
- **y** (float) – The height of the peak in the EIC.
- **charge** (int) – The charge on the peak. Default 0.
- **label** (str) – The label of the peak. e.g. "M+H" . Default ''.

Attributes:

charge	The charge on the peak.
label	The label of the peak.
rx	
x	
y	

Methods:

from_dict(d)	Construct an instance of *Peak* from a dictionary.
from_xml(element)	Construct a *Peak* object from an XML element.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Peak* object.

charge
Type: int

The charge on the peak.

classmethod from_dict(*d*)
Construct an instance of *Peak* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

classmethod from_xml(*element*)
Construct a *Peak* object from an XML element.

Parameters **element** (ObjectifiedElement) – a <p> XML element from an <MSPeaks> element of a CEF file

Return type *Peak*

label
Type: str

The label of the peak. e.g. "M+H"

rx
Type: float

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *Peak* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

x
Type: float

y
Type: float

The height of the peak in the EIC.

class RTRange(*start=0.0*, *end=0.0*)
Bases: object

Represents an <RTRange> element from a CEF file.

Parameters

- **start** (Union[float, timedelta]) – The start time in minutes . Default 0.0.
- **end** (Union[float, timedelta]) – The end time in minutes . Default 0.0.

Attributes:

end	The end time in minutes
start	The start time in minutes

Methods:

from_dict(d)	Construct an instance of *RTRange* from a dictionary.
from_xml(element)	Construct ab *RTRange* object from an XML element.
to_dict([convert_values])	Returns a dictionary containing the contents of the *RTRange* object.

end
Type: timedelta

The end time in minutes

classmethod from_dict(*d*)
Construct an instance of *RTRange* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

classmethod from_xml(*element*)

Construct ab *RTRange* object from an XML element.

Parameters **element** (ObjectifiedElement) – The <RTRange> XML element to parse the data from.

Return type *RTRange*

start
Type: timedelta

The start time in minutes

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *RTRange* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

class Score(*score*, *flag_string=''*, *flag_severity=0*)
Bases: float

A score indicating how well the compound matches the observed spectrum.

Parameters

- **score** – The score
- **flag_string** (str) – Optional flag. See *Flag* for details. Default ''.
- **flag_severity** (int) – The severity of the flag. Default 0.

class Spectrum(*spectrum_type=''*, *algorithm=''*, *saturation_limit=0*, *scans=0*, *scan_type=''*, *ionisation=''*, *polarity=0*, *voltage=0.0*, *device=None*, *peaks=None*, *rt_ranges=None*)
Bases: Dictable

Agilent CEF Spectrum.

Parameters

- **spectrum_type** (str) – The type of spectrum e.g. 'FbF'. Default ''.
- **algorithm** (str) – The algorithm used to identify the compound. Default ''.
- **saturation_limit** (int) – Unknown. Might mean saturation limit?. Default 0.
- **scans** (int) – Unknown. Presumably the number of scans that make up the spectrum?. Default 0.
- **scan_type** (str) – Default ''.
- **ionisation** (str) – The type of ionisation e.g. ESI. Default ''.
- **polarity** (Union[str, int]) – The polarity of the ionisation. Default 0.
- **device** (Optional[*Device*]) – The device that acquired the data. Default None.
- **peaks** (Optional[Sequence[*Peak*]]) – A list of identified peaks in the mass spectrum. Default None.
- **rt_ranges** (Optional[Sequence[*RTRange*]]) – A list of retention time ranges for the mass spectrum. Default None.

classmethod from_xml(*element*)
Construct a *Spectrum* object from an XML element.

Parameters **element** (ObjectifiedElement) – a Spectrum XML element from a CEF file

Return type *Spectrum*

Chapter

FIVE

mh_utils.csv_parser

Parser for CSV result files produced by MassHunter Qualitative.

New in version 0.2.0.

ResultParser(raw_results_dir, ...)	Given a directory of CSV results exported from MassHunter, parse them to CSV and JSON.
parse_masshunter_csv(csv_file, csv_outfile, ...)	Parse CSV results files created by MassHunter.

class ResultParser (*raw_results_dir*, *json_results_dir*, *csv_results_dir*)

Bases: object

Given a directory of CSV results exported from MassHunter, parse them to CSV and JSON.

Parameters

- **raw_results_dir** (Union[str, Path, PathLike]) – The directory in which the raw exports from MassHunter are stored.
- **json_results_dir** (Union[str, Path, PathLike]) – The directory to store the output json files in.
- **csv_results_dir** (Union[str, Path, PathLike]) – The directory to store the output csv files in.

parse_directory_list (*directory_list*)

Runs parse_for_directory() for each directory in directory_list.

Parameters **directory_list** (Iterable[Union[str, Path, PathLike]]) – A list of directories to process.

parse_for_directory (*directory*)

Convert the "CSV Results.csv" file in the given directory to CSV and JSON.

Parameters **directory** (Union[str, Path, PathLike])

parse_masshunter_csv (*csv_file*, *csv_outfile*, *json_outfile*)

Parse CSV results files created by MassHunter.

Parameters

- **csv_file** (Union[str, Path, PathLike])
- **csv_outfile** (Union[str, Path, PathLike])
- **json_outfile** (Union[str, Path, PathLike])

5.3 mh_utils.csv_parser.classes

Classes to model parts of MassHunter CSV files.

New in version 0.2.0.

Classes:

BaseSamplePropertyDict	OrderedDict to store a single property of a set of samples.
Result(cas, name, hits[, index, formula, ...])	Represents a Result in a MassHunter CSV file.
Sample(sample_name, sample_type, ...[, results])	Represents a sample in a MassHunter CSV file.
SampleList([iterable])	A list of *mh_utils.csv_parser.classes.Sample* objects.
SamplesAreaDict	collections.OrderedDict to store area information parsed from MassHunter results CSV files.
SamplesScoresDict	collections.OrderedDict to store score information parsed from MassHunter results CSV files.

Data:

_R	Invariant TypeVar bound to *mh_utils.csv_parser.classes.Result*.
_S	Invariant TypeVar bound to *mh_utils.csv_parser.classes.Sample*.
_SL	Invariant TypeVar bound to *mh_utils.csv_parser.classes.SampleList*.

class BaseSamplePropertyDict

Bases: OrderedDict

OrderedDict to store a single property of a set of samples.

Keys are the sample names and the values are dictionaries mapping compound names to property values.

Attributes:

n_compounds	Returns the number of compounds in the *BaseSamplePropertyDict*.
n_samples	Returns the number of samples in the *BaseSamplePropertyDict*.
sample_names	Returns a list of sample names in the *BaseSamplePropertyDict*.

property n_compounds

Returns the number of compounds in the *BaseSamplePropertyDict*.

Return type int

property n_samples

Returns the number of samples in the *BaseSamplePropertyDict*.

Return type int

property sample_names
Returns a list of sample names in the *BaseSamplePropertyDict*.

Return type List[str]

class Result(*cas*, *name*, *hits*, *index=- 1*, *formula=''*, *score=0.0*, *abundance=0*, *height=0*, *area=0*, *diff_mDa=0.0*, *diff_ppm=0.0*, *rt=0.0*, *start=0.0*, *end=0.0*, *width=0.0*, *tgt_rt=0.0*, *rt_diff=0.0*, *mz=0.0*, *product_mz=0.0*, *base_peak=0.0*, *mass=0.0*, *average_mass=0.0*, *tgt_mass=0.0*, *mining_algorithm=''*, *z_count=0*, *max_z=0*, *min_z=0*, *n_ions=0*, *polarity=''*, *label=''*, *flags=''*, *flag_severity=''*, *flag_severity_code=0*)
Bases: Dictable

Represents a Result in a MassHunter CSV file.

Parameters

- **cas**
- **name** (str)
- **hits**
- **index** (int) – Default -1.
- **formula** (str) – Default ''.
- **score** (float) – Default 0.0.
- **abundance** (float) – Default 0.
- **height** (float) – Default 0.
- **area** (float) – Default 0.
- **diff_mDa** (float) – Default 0.0.
- **diff_ppm** (float) – Default 0.0.
- **rt** (float) – Default 0.0.
- **start** (float) – Default 0.0.
- **end** (float) – Default 0.0.
- **width** (float) – Default 0.0.
- **tgt_rt** (float) – Default 0.0.
- **rt_diff** (float) – Default 0.0.
- **mz** (float) – Default 0.0.
- **product_mz** (float) – Default 0.0.
- **base_peak** (float) – Default 0.0.
- **mass** (float) – Default 0.0.
- **average_mass** (float) – Default 0.0.
- **tgt_mass** (float) – Default 0.0.
- **mining_algorithm** (str) – Default ''.
- **z_count** (int) – Default 0.
- **max_z** (int) – Default 0.
- **min_z** (int) – Default 0.
- **n_ions** (int) – Default 0.
- **polarity** (str) – Default ''.
- **label** (str) – Default ''.
- **flags** (str) – Default ''.
- **flag_severity** (str) – Default ''.
- **flag_severity_code** (int) – Default 0.

Methods:

from_series(series)	Consruct a *Result* from a pandas.Series.

classmethod from_series(*series*)
Consruct a *Result* from a pandas.Series.

Parameters **series** (Series)

Return type *~_R*

class Sample(*sample_name*, *sample_type*, *instrument_name*, *position*, *user*, *acq_method*, *da_method*, *irm_cal_status*, *filename*, *results=None*)

Bases: Dictable

Represents a sample in a MassHunter CSV file.

Parameters

- **sample_name**
- **sample_type**
- **instrument_name**
- **position**
- **user**
- **acq_method**
- **da_method**
- **irm_cal_status**
- **filename**
- **results** – Default None.

Methods:

add_result(result)	Add a result to the sample.
from_series(series)	Constuct a *Sample* from a pandas.Series.

Attributes:

results_list	Returns a list of results in the order in which they were identified.

add_result(*result*)

Add a result to the sample.

Parameters **result**

classmethod from_series(*series*)

Constuct a *Sample* from a pandas.Series.

Parameters **series**

Return type *~_S*

Returns

property results_list

Returns a list of results in the order in which they were identified.

I.e. sorted by the Cpd value from the csv export.

Return type List[*Result*]

class SampleList(*iterable=()*, */*)

Bases: List[*Sample*]

A list of *mh_utils.csv_parser.classes.Sample* objects.

Methods:

add_new_sample(*args, **kwargs)	Add a new sample to the list and return the Sample object representing it.
add_sample(sample)	Add a *Sample* object to the list.
add_sample_from_series(series)	Create a new sample object from a pandas.series and add it to the list.
filter(sample_names[, key, exclude])	Filter the list to only contain sample_names whose name is in sample_names.
from_json_file(filename, **kwargs)	Construct a *SampleList* from JSON file.
get_areas_and_scores(compound_name[, ...])	Returns two dictionaries: one containing sample names and peak areas for the compound with the given name, the other containing sample names and scores.
get_areas_and_scores_for_compounds(...[, ...])	Returns two dictionaries: one containing sample names and peak areas for the compounds with the given names, the other containing sample names and scores.
get_areas_for_compounds(compound_names[, ...])	Returns a dictionary containing sample names and peak areas for the compounds with the given names.
get_compounds()	Returns a list containing the names of the compounds present in the samples in alphabetical order.
get_peak_areas(compound_name[, include_none])	Returns a dictionary containing sample names and peak areas for the compound with the given name.
get_retention_times(compound_name[, ...])	Returns a dictionary containing sample names and retention times for the compound with the given name.
get_scores(compound_name[, include_none])	Returns a dictionary containing sample names and scores for the compound with the given name.
rename_samples(rename_mapping[, key])	Rename the samples in the list.
reorder_samples(order_mapping[, key])	Reorder the list of Samples in place.
sort_samples(key[, reverse])	Sort the list of Samples in place.

Attributes:

sample_names	Returns a list of sample names in the *SampleList*.

add_new_sample(**args*, ***kwargs*)

Add a new sample to the list and return the Sample object representing it.

add_sample(*sample*)

Add a *Sample* object to the list.

Parameters **sample** (*Sample*)

Return type *Sample*

add_sample_from_series(*series*)
Create a new sample object from a `pandas.series` and add it to the list.

Return type *Sample*

Returns The newly created `Sample` object.

Parameters **series** (`Series`)

filter(*sample_names*, *key='sample_name'*, *exclude=False*)
Filter the list to only contain sample_names whose name is in `sample_names`.

Parameters

- **sample_names** (`Iterable[str]`) – A list of sample names to include
- **key** (`str`) – The name of the property in the sample to sort by. Default `'sample_name'`.
- **exclude** (`bool`) – If `True`, any sample whose name is in `sample_names` will be excluded from the output, rather than included. Default `False`.

Return type *~_SL*

classmethod from_json_file(*filename*, ***kwargs*)
Construct a *SampleList* from JSON file.

Parameters

- **filename** (`Union[str, Path, PathLike]`) – The filename of the JSON file.
- ****kwargs** – Keyword arguments passed to `domdf_python_tools.paths.PathPlus.load_json()`.

Return type *~_SL*

get_areas_and_scores(*compound_name*, *include_none=False*)
Returns two dictionaries: one containing sample names and peak areas for the compound with the given name, the other containing sample names and scores.

Parameters

- **compound_name** (`str`)
- **include_none** (`bool`) – Whether samples where the compound was not found should be included in the results. Default `False`.

Return type `Tuple[OrderedDict, OrderedDict]`

get_areas_and_scores_for_compounds(*compound_names*, *include_none=False*)
Returns two dictionaries: one containing sample names and peak areas for the compounds with the given names, the other containing sample names and scores.

Parameters

- **compound_names** (`Iterable[str]`)
- **include_none** (`bool`) – Whether samples where none of the specified compounds were found should be included in the results. Default `False`.

Return type `Tuple[SamplesAreaDict, SamplesScoresDict]`

get_areas_for_compounds(*compound_names*, *include_none=False*)
Returns a dictionary containing sample names and peak areas for the compounds with the given names.

Parameters

- **compound_names** (Iterable[str])
- **include_none** (bool) – Whether samples where none of the specified compounds were found should be included in the results. Default False.

Return type *SamplesAreaDict*

get_compounds()
Returns a list containing the names of the compounds present in the samples in alphabetical order.

Return type List[str]

get_peak_areas(*compound_name*, *include_none=False*)
Returns a dictionary containing sample names and peak areas for the compound with the given name.

Parameters

- **compound_name** (str)
- **include_none** (bool) – Whether samples where the compound was not found should be included in the results. Default False.

Return type OrderedDict

get_retention_times(*compound_name*, *include_none=False*)
Returns a dictionary containing sample names and retention times for the compound with the given name.

Parameters

- **compound_name** (str)
- **include_none** (bool) – Whether samples where the compound was not found should be included in the results. Default False.

Return type OrderedDict

get_scores(*compound_name*, *include_none=False*)
Returns a dictionary containing sample names and scores for the compound with the given name.

Parameters

- **compound_name** (str)
- **include_none** (bool) – Whether samples where the compound was not found should be included in the results. Default False.

Return type OrderedDict

rename_samples(*rename_mapping*, *key='sample_name'*)
Rename the samples in the list.

Parameters

- **rename_mapping** (Dict) – A mapping between current sample names and their new names.
- **key** (str) – The name of the property in the sample to sort by. Default 'sample_name'.

Use rename_mapping=:py:obj:*None* or omit the sample from the rename_mapping entirely to leave the name unchanged.

For example:

```
rename_mapping = {
    "Propellant 1ug +ve": "Alliant Unique 1µg/L +ESI",
    "Propellant 1mg +ve": "Alliant Unique 1mg/L +ESI",
    "Propellant 1mg -ve": None,
    }
```

reorder_samples(*order_mapping*, *key='sample_name'*)
Reorder the list of Samples in place.

Parameters

- **order_mapping** (Dict) – A mapping between sample names and their new position in the list. For example:

```
order_mapping = {
    "Propellant 1ug +ve": 0,
    "Propellant 1mg +ve": 1,
    "Propellant 1ug -ve": 2,
    "Propellant 1mg -ve": 3,
    }
```

- **key** (str) – The name of the property in the sample to sort by. Default 'sample_name'.

property sample_names
Returns a list of sample names in the *SampleList*.

Return type List[str]

sort_samples(*key*, *reverse=False*)
Sort the list of Samples in place.

Parameters

- **key** (str) – The name of the property in the sample to sort by.
- **reverse** (bool) – Whether the list should be sorted in reverse order. Default False.

Return type

class SamplesAreaDict

Bases: *BaseSamplePropertyDict*

collections.OrderedDict to store area information parsed from MassHunter results CSV files.

Methods:

get_compound_areas(compound_name)	Get the peak areas for the given compound in every sample.

get_compound_areas(*compound_name*)

Get the peak areas for the given compound in every sample.

Parameters **compound_name** (str)

Return type List[float]

class SamplesScoresDict

Bases: *BaseSamplePropertyDict*

collections.OrderedDict to store score information parsed from MassHunter results CSV files.

Methods:

get_compound_scores(compound_name)	Get the peak scores for the given compound in every sample.

get_compound_scores(*compound_name*)

Get the peak scores for the given compound in every sample.

Parameters **compound_name** (str)

Return type List[float]

_R = TypeVar(_R, bound=Result)

Type: TypeVar

Invariant TypeVar bound to *mh_utils.csv_parser.classes.Result*.

_S = TypeVar(_S, bound=Sample)

Type: TypeVar

Invariant TypeVar bound to *mh_utils.csv_parser.classes.Sample*.

_SL = TypeVar(_SL, bound=SampleList)

Type: TypeVar

Invariant TypeVar bound to *mh_utils.csv_parser.classes.SampleList*.

5.4 mh_utils.csv_parser.utils

CSV utility functions.

New in version 0.2.0.

Functions:

concatenate_json(*files[, outfile])	Concatenate multiple JSON files together and return a list of `Sample` objects in the concatenated json output.
drop_columns(df, *[, axis, inplace])	Drop columns from the MassHunter CSV file.
reorder_columns(df)	Reorder columns from the MassHunter CSV file.

concatenate_json(**files*, *outfile=None*)

Concatenate multiple JSON files together and return a list of `Sample` objects in the concatenated json output.

Parameters

- ***files** (Union[str, Path, PathLike]) – The files to concatenate.
- **outfile** (Union[str, Path, PathLike, None]) – The file to save the output as. If `None` no file will be saved. Default `None`.

Return type *SampleList*

drop_columns(*df*, ***, *axis=1*, *inplace=True*, ***kwargs*)

Drop columns from the MassHunter CSV file.

Parameters

- **df** (DataFrame) – The `pandas.DataFrame` to drop columns in.
- **axis** (int) – Which axis to drop columns on. Default `1`.
- **inplace** (bool) – Whether to modify the `pandas.DataFrame` in place. Default `True`.
- **kwargs** – Additional keyword arguments passed to `pandas.DataFrame.drop()`.

Return type DataFrame

reorder_columns(*df*)

Reorder columns from the MassHunter CSV file.

Parameters **df** (DataFrame) – The `pandas.DataFrame` to reorder columns in.

Return type DataFrame

Chapter

SIX

mh_utils.worklist_parser

Parser for MassHunter worklists.

Only one function is defined here: *read_worklist()*, which reads the reads the given worklist file and returns a *mh_utils.worklist_parser.classes.Worklist* file representing it. The other functions and classes must be imported from submodules of this package.

read_worklist (*filename*)
Read the worklist from the given file.

Parameters **filename** (Union[str, Path, PathLike]) – The filename of the worklist.

Return type *Worklist*

6.2 mh_utils.worklist_parser.classes

Main classes for the worklist parser.

Classes:

Attribute(attribute_id, attribute_type, ...)	Represents an Attribute.
Checksum(SchemaVersion, ALGO_VERSION, HASHCODE)	Represents a checksum for a worklist.
JobData(id, job_type, run_status[, sample_info])	Represents an entry in the worklist.
Macro(project_name, procedure_name, ...)	Represents a macro in a worklist.
Worklist(version, locked_run_mode, ...)	Class that represents an Agilent MassHunter worklist.

class Attribute (*attribute_id*, *attribute_type*, *field_type*, *system_name*, *header_name*, *data_type*, *default_data_value*, *reorder_id*, *show_hide_status*, *column_width*)
Represents an Attribute.

Parameters

- **attribute_id** (int)
- **attribute_type** (*AttributeType*) – The attribute type identifier.
- **field_type** (int) – The field type identifier.
- **system_name** (str)
- **header_name** (str)
- **data_type** (int)
- **default_data_value** (str)
- **reorder_id** (int)
- **show_hide_status** (Union[str, bool])
- **column_width** (int) – end{multicols}

Attributes:

attribute_id	
attribute_type	The attribute type identifier.
column_width	
data_type	
default_data_value	
field_type	The field type identifier.
header_name	
reorder_id	
show_hide_status	
system_name	

Methods:

from_dict(d)	Construct an instance of *Attribute* from a dictionary.
from_xml(element)	Construct an *Attribute* object from an XML element.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Attribute* object.

attribute_id
Type: int

attribute_type
Type: *AttributeType*

The attribute type identifier.

Can be System Defined (0), System Used (1), or User Added (2).

column_width
Type: int

data_type
Type: Any

default_data_value
Type: str

field_type
Type: int

The field type identifier.

Each of the system defined columns have a field type starting from sampleid = 0 to reserved6 = 24.

The system used column can be ‘compound param’ = 35, ‘optim param’ = 36, ‘mass param’ = 37 and ‘protein param’ = 38.

The User added columns start from 45.

classmethod from_dict(*d*)
Construct an instance of *Attribute* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

classmethod from_xml(*element*)
Construct an *Attribute* object from an XML element.

Return type *Attribute*

header_name
Type: str

reorder_id
Type: int

show_hide_status
Type: bool

system_name
Type: str

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *Attribute* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

class Checksum(*SchemaVersion*, *ALGO_VERSION*, *HASHCODE*)
Represents a checksum for a worklist.

The format of the checksum is unknown.

Parameters

- **SchemaVersion** (int)
- **ALGO_VERSION** (int)
- **HASHCODE** (str)

Attributes:

ALGO_VERSION
HASHCODE
SchemaVersion

Methods:

from_dict(d)	Construct an instance of *Checksum* from a dictionary.
from_xml(element)	Construct a *Checksum* object from an XML element.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Checksum* object.

ALGO_VERSION
Type: int

HASHCODE
Type: str

SchemaVersion
Type: int

classmethod from_dict(*d*)
Construct an instance of *Checksum* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

classmethod from_xml(*element*)
Construct a *Checksum* object from an XML element.

Return type *Checksum*

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *Checksum* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

class JobData(*id*, *job_type*, *run_status*, *sample_info=None*)
Bases: Dictable

Represents an entry in the worklist.

Parameters

- **id** (Union[str, UUID]) – The ID of the job.
- **job_type** (int) – The type of job. TODO: enum of values
- **run_status** (int) – The status of the analysis. TODO: enum of values
- **sample_info** (Optional[dict]) – Optional `key: value` mapping of information about the sample. Default None.

Methods:

from_xml(element[, user_columns])	Construct a *JobData* object from an XML element.

classmethod from_xml(*element*, *user_columns=None*)
Construct a *JobData* object from an XML element.

Parameters

- **element** (ObjectifiedElement) – The XML element to parse the data from
- **user_columns** (Optional[Dict[str, *Column*]]) – Optional mapping of user column labels to *Column* objects. Default None.

Return type *JobData*

class Macro(*project_name*, *procedure_name*, *input_parameter*, *output_data_type*, *output_parameter*, *display_string*)

Represents a macro in a worklist.

Parameters

- **project_name** (str)
- **procedure_name** (str)
- **input_parameter** (str)
- **output_data_type** (int)
- **output_parameter** (str) – .
- **display_string** (str)

Attributes:

display_string	
input_parameter	
output_data_type	
output_parameter	
procedure_name	
project_name	
undefined	Returns whether the macro is undefined.

Methods:

from_dict(d)	Construct an instance of *Macro* from a dictionary.
from_xml(element)	Construct a *Macro* object from an XML element.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Macro* object.

display_string

Type: str

classmethod from_dict(*d*)

Construct an instance of *Macro* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

classmethod from_xml(*element*)

Construct a *Macro* object from an XML element.

Return type *Macro*

input_parameter

Type: str

output_data_type

Type: int

output_parameter

Type: str

procedure_name
Type: str

project_name
Type: str

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *Macro* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

property undefined
Returns whether the macro is undefined.

Return type bool

class Worklist(*version*, *locked_run_mode*, *instrument_name*, *params*, *user_columns*, *jobs*, *checksum*)
Bases: *XMLFileMixin*, Dictable

Class that represents an Agilent MassHunter worklist.

Parameters

- **version** (float) – WorklistInfo version number
- **locked_run_mode** (bool) – Flag to indicate whether the data was acquired in locked mode. Yes = -1. No = 0.
- **instrument_name** (str) – The name of the instrument.
- **params** (dict) – Mapping of parameter names to values. TODO: Check
- **user_columns** (dict) – Mapping of user columns to ??? TODO
- **jobs** (Sequence[*JobData*])
- **checksum** (*Checksum*) – The checksum of the worklist file. The format is unknown.

Methods:

__repr__()	Return a string representation of the *Worklist*.
as_dataframe()	Returns the *Worklist* as a pandas.DataFrame.
from_xml(element)	Construct a *Worklist* object from an XML element.

__repr__()
Return a string representation of the *Worklist*.

Return type str

as_dataframe()
Returns the *Worklist* as a pandas.DataFrame.

Return type DataFrame

classmethod from_xml(*element*)
Construct a *Worklist* object from an XML element.

Return type *Worklist*

6.3 mh_utils.worklist_parser.columns

Properties for columns in a Worklist.

Column(name, attribute_id, attribute_type, ...)	Represents a column in a worklist.
injection_volume(val)	Handle special case for injection volume of -1, which indicates "As Method".
columns	Mapping of column names to column objects.

class Column(*name*, *attribute_id*, *attribute_type*, *dtype*, *default_value*, *field_type=None*, *reorder_id=None*)
Bases: object

Represents a column in a worklist.

Parameters

- **name** (str) – The name of the column
- **attribute_id** (int)
- **attribute_type** (*AttributeType*) – The attribute type identifier.
- **dtype** (Callable) – The field datatype.
- **default_value** (Any)
- **field_type** (Optional[int]) – The field type identifier. Default None.
- **reorder_id** (Optional[int]) – Default None.

attribute_id
Type: int

attribute_type
Type: *AttributeType*

The attribute type identifier.

Can be System Defined (0), System Used (1), or User Added (2).

cast_value(*value*)
Cast value to the dtype of this column.

default_value
Type: Any

dtype
Type: Callable

field_type
Type: Optional[int]

The field type identifier.

Each of the system defined columns have a field type starting from sampleid = 0 to reserved6 = 24.

The system used column can be 'compound param' = 35, 'optim param' = 36, 'mass param' = 37 and 'protein param' = 38.

The User added columns start from 45.

classmethod from_attribute(*attribute*)
Construct a column for a *mh_utils.worklist_parser.classes.Attribute*.

Parameters **attribute** (*Attribute*) – *Attribute*

Return type *Column*

classmethod from_dict(*d*)
Construct an instance of *Column* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

name
Type: str

The name of the column

reorder_id
Type: Optional[int]

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *Column* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

injection_volume(*val*)
Handle special case for injection volume of -1, which indicates "As Method".

Parameters **val** (Union[float, str])

Returns

Return type Union[int, str]

columns
Mapping of column names to column objects.

6.4 mh_utils.worklist_parser.enums

Enumerations of values.

Classes:

AttributeType(value)	Enumeration of values for column/attribute types.

enum AttributeType(*value*)

Bases: enum_tools.custom_enums.IntEnum

Enumeration of values for column/attribute types.

Member Type int

Valid values are as follows:

SystemDefined = <AttributeType.SystemDefined: 0>

SystemUsed = <AttributeType.SystemUsed: 1>

UserAdded = <AttributeType.UserAdded: 2>

6.5 mh_utils.worklist_parser.parser

MassHunter worklist parser.

Functions:

parse_datetime(the_date)	Parse a datetime from a worklist or contents file.
parse_params(element)	Parse the worklist execution parameters from XML.
parse_sample_info(element[, user_columns])	Parse information about a sample in a worklist from XML.

Data:

sample_info_tags	Mapping of XML tag names to attribute names.

parse_datetime(*the_date*)

Parse a datetime from a worklist or contents file.

Parameters **the_date** (str) – The date and time as a string in the format %Y-%m-%dT%H:%M:%S%z.

Return type datetime

parse_params(*element*)

Parse the worklist execution parameters from XML.

Parameters **element** (ObjectifiedElement)

Return type Dict[str, Any]

Returns Mapping of keys to parameter values.

parse_sample_info(*element*, *user_columns=None*)
Parse information about a sample in a worklist from XML.

Parameters

- **element** (ObjectifiedElement) – The XML element to parse the data from
- **user_columns** (Optional[Dict[str, *Column*]]) – Optional mapping of user column labels to *Column* objects. Default None.

Return type Dict[str, Any]

sample_info_tags
Mapping of XML tag names to attribute names.

6.6 Example Usage

Listing 1: read_worklist.py

```python
# 3rd party
from pandas import DataFrame

# this package
from mh_utils.worklist_parser import Worklist, read_worklist

# Replace 'worklist.wkl' with the filename of your worklist.
wkl: Worklist = read_worklist("worklist.wkl")

df: DataFrame = wkl.as_dataframe()

# Filter columns
df = df[["Sample Name", "Method", "Data File", "Drying Gas", "Gas Temp",
↪"Nebulizer"]]

# Get just the filename from 'Method' and 'Data File'
df["Method"] = [x.name for x in df["Method"]]
df["Data File"] = [x.name for x in df["Data File"]]

# Show the DataFrame
print(df.to_string())
```

Listing 2: worklist_df_head.txt

```
                  Sample Name                                        Method
↪                           Data File   Drying Gas   Gas Temp   Nebulizer
0           Methanol Blank +ve        Maitre Gunshot Residue Positive.m
↪Methanol_Blank_+ve_191121-0001-r001.d
1           Propellant 1mg +ve        Maitre Gunshot Residue Positive.m
↪Propellant_1mg_+ve_191121-0002-r001.d
2           Propellant 1ug +ve        Maitre Gunshot Residue Positive.m
↪Propellant_1ug_+ve_191121-0003-r001.d
3           Methanol Blank -ve        Maitre Gunshot Residue Negative.m
↪Methanol_Blank_-ve_191121-0004-r001.d
```

(continues on next page)

(continued from previous page)

```
4           Propellant 1mg -ve        Maitre Gunshot Residue Negative.m
↪Propellant_1mg_-ve_191121-0005-r001.d
5           Propellant 1ug -ve        Maitre Gunshot Residue Negative.m
↪Propellant_1ug_-ve_191121-0006-r001.d
6       Methanol Blank +ve 5ul    Maitre Gunshot Residue Positive 5ul.m
↪Methanol_Blank_+ve_5ul_191121-0007-r001.d
7       Propellant 1mg +ve 5ul    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_+ve_5ul_191121-0008-r001.d
8       Propellant 1ug +ve 5ul    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_+ve_5ul_191121-0009-r001.d
9               Methanol Blank    Maitre Gunshot Residue Positive 5ul.m
↪   Methanol_Blank_191121-0010-r001.d
10      Propellant 1mg gas 200    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_gas_200_191121-0011-r001.d                  200
11      Propellant 1ug gas 200    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_gas_200_191121-0012-r001.d                  200
12      Propellant 1mg gas 280    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_gas_280_191121-0013-r001.d                  280
13      Propellant 1ug gas 280    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_gas_280_191121-0014-r001.d                  280
14    Propellant 1mg drying 14    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_drying_14_191121-0015-r001.d          14
15    Propellant 1ug drying 14    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_drying_14_191121-0016-r001.d          14
16    Propellant 1mg drying 16    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_drying_16_191121-0017-r001.d          16
17    Propellant 1ug drying 16    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_drying_16_191121-0018-r001.d          16
18    Propellant 1mg drying 18    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_drying_18_191121-0019-r001.d          18
19    Propellant 1ug drying 18    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_drying_18_191121-0020-r001.d          18
20     Propellant 1mg nebul 40    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_nebul_40_191121-0021-r001.d                  40
21     Propellant 1ug nebul 40    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_nebul_40_191121-0022-r001.d                  40
22     Propellant 1mg nebul 50    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_nebul_50_191121-0023-r001.d                  50
23     Propellant 1ug nebul 50    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_nebul_50_191121-0024-r001.d                  50
24     Propellant 1mg nebul 60    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1mg_nebul_60_191121-0025-r001.d                  60
25     Propellant 1ug nebul 60    Maitre Gunshot Residue Positive 5ul.m
↪Propellant_1ug_nebul_60_191121-0026-r001.d                  60
...
```

Listing 3: read_worklist.py

```
21
22 # save as CSV
23 df.to_csv("worklist.csv")
24
25 # save as JSON
26 df.to_json("worklist.json", indent=2)
```

6.6.1 Output

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

S

T

U

W

X

Y