mh-utils

Release 0.2.2

Dominic Davis-Foster

Mar 17, 2021

CONTENTS

Utilities for handing ancillary files produced by MassHunter.

The current utilities are as follows:

- `mh_utils.worklist_parser`: Parse Agilent MassHunter Worklists (`*.wkl` files).
- `mh_utils.cef_parser`: Parse Agilent MassHunter Compound Exchange Format files (`*.cef` 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/domdfcoding/mh_utils@master --user
```

CHAPTER

TWO

API REFERENCE

2.1 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`

2.2 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`.

2.3 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 dictionary mapping algorithms to scores.
parse_match_scores(element)	Parse a <MatchScores> element into a dictionary mapping 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** – String identifying the type of device.
- **number**

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

__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

__repr__()

Returns a string representation of the *Flag*.

Return type str

typeddict LocationDict

Bases: dict

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 dictionary mapping 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 dictionary mapping 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**
- **rx**
- **y** – The height of the peak in the EIC.
- **charge** – The charge on the peak. Default 0.

- **label** – 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 <MS-Peaks> 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** – The start time in minutes . Default 0.0.
- **end** – 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.

Methods:

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

__repr__()

Returns a string representation of the *Spectrum*.

Return type str

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*

2.4 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.

Functions:

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

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

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

Return type *Worklist*

2.4.1 mh_utils.worklist_parser.classes

Main classes for the worklist paser.

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])	Class that 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*)
Bases: object

Represents an Attribute.

Parameters

- **attribute_id**
- **attribute_type** – Can be System Defined (0), System Used (1), or User Added (2).
- **field_type** – 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.
- **system_name**
- **header_name**
- **data_type**
- **default_data_value**
- **reorder_id**

- **show_hide_status**
- **column_width**

Attributes:

attribute_id	
attribute_type	Can be System Defined (0), System Used (1), or User Added (2).
column_width	
data_type	
default_data_value	
field_type	Each of the system defined columns have a field type starting from sampleid = 0 to reserved6 = 24.
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*

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

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*)
Bases: object

Represents a checksum for a worklist.

The format of the checksum is unknown.

Parameters

- **SchemaVersion**
- **ALGO_VERSION**
- **HASHCODE**

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

Class that 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:

__repr__()	Return a string representation of the Dictable.
from_xml(element[, user_columns])	Construct a *JobData* object from an XML element.

__repr__()
Return a string representation of the Dictable.

Return type str

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*)
Bases: object

Represents a macro in a worklist.

Parameters

- **project_name**
- **procedure_name**
- **input_parameter**
- **output_data_type**
- **output_parameter**
- **display_string**

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 repr(self).
as_dataframe()	Returns the *Worklist* as a pandas.DataFrame.
from_xml(element)	Construct a *Worklist* object from an XML element.

__repr__()
Return repr(self).

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*

2.4.2 mh_utils.worklist_parser.columns

Properties for columns in a Worklist.

Classes:

Column(name, attribute_id, attribute_type, ...)	Represents a column in a worklist.

Functions:

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

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** – The name of the column
- **attribute_id**
- **attribute_type** – can be System Defined = 0, System Used = 1, User Added = 2
- **dtype** (Callable)
- **default_value** (Any)
- **field_type** (Optional[int]) – 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. Default None.
- **reorder_id** (Optional[int]) – Default None.

Methods:

__eq__(other)	Return self == other.
__ge__(other)	Return self >= other.
__getstate__()	Used for pickling.
__gt__(other)	Return self > other.
__le__(other)	Return self <= other.
__lt__(other)	Return self < other.
__ne__(other)	Return self != other.
__repr__()	Return a string representation of the *Column*.
__setstate__(state)	Used for pickling.

cast_value(value)	Cast value to the dtype of this column.
from_attribute(attribute)	Construct a column for an *Attribute*.
from_dict(d)	Construct an instance of *Column* from a dictionary.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Column* object.

Attributes:

attribute_id	
attribute_type	can be System Defined = 0, System Used = 1, User Added = 2
default_value	
dtype	
field_type	Each of the system defined columns have a field type starting from sampleid = 0 to reserved6 = 24.
name	The name of the column
reorder_id	

__eq__(*other*)
Return `self == other`.

Return type bool

__ge__(*other*)
Return `self >= other`.

Return type bool

__getstate__()
Used for pickling.

Automatically created by attrs.

__gt__(*other*)
Return `self > other`.

Return type bool

__le__(*other*)
Return `self <= other`.

Return type bool

__lt__(*other*)
Return `self < other`.

Return type bool

__ne__(*other*)
Return `self != other`.

Return type bool

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

Return type str

__setstate__(*state*)
Used for pickling.

Automatically created by attrs.

attribute_id
Type: int

attribute_type
Type: *AttributeType*

can be System Defined = 0, System Used = 1, 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]`

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 an *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.

2.4.3 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>
Attributes defined by the system.

SystemUsed = <AttributeType.SystemUsed: 1>
Attributes used by the system.

UserAdded = <AttributeType.UserAdded: 2>
Attributes added by the user.

2.4.4 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 following 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.

2.4.5 Example Usage

read_worklist.py worklist.xml

```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
```

```
                  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
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
```

(continues on next page)

(continued from previous page)

```
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
...
```

```
print(df.to_string())

# save as CSV
df.to_csv("worklist.csv")

# save as JSON
df.to_json("worklist.json", indent=2)
```

Output

```
worklist.csv worklist.json
```

CHAPTER

THREE

DOCUMENTATION

3.1 Overview

`mh_utils` uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

3.2 Coding style

formate is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

3.3 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

3.4 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

3.5 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

3.6 Downloading source code

The `mh_utils` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/mh_utils

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/mh_utils"
> Cloning into 'mh_utils'...
> remote: Enumerating objects: 47, done.
> remote: Counting objects: 100% (47/47), done.
> remote: Compressing objects: 100% (41/41), done.
> remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
> Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
> Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

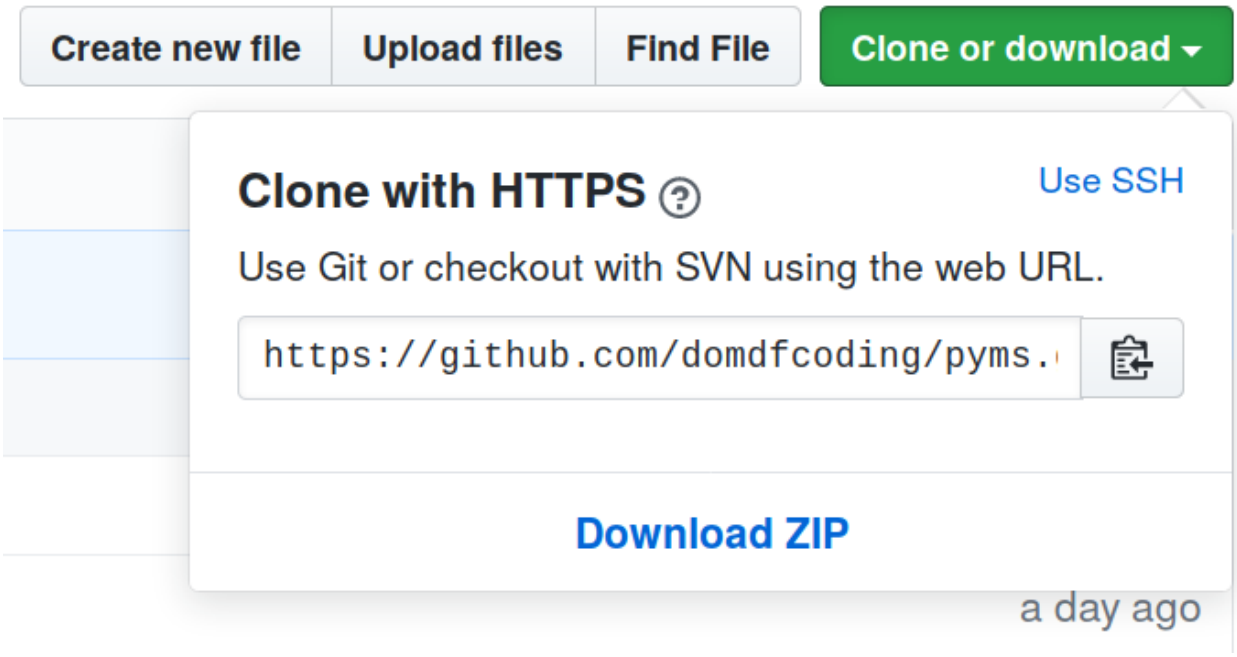

Fig. 1: Downloading a 'zip' file of the source code

3.6.1 Building from source

The recommended way to build `mh_utils` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

PYTHON MODULE INDEX

m

PYTHON MODULE INDEX

m

INDEX

Symbols

A

C

D

E

F

W

X

Y