pyHMSA is a pure Python implementation of the MSA / MAS / AMAS HyperDimensional Data File (HMSA, for short) specifications. This file format is intended to be a common exchange format for microscopy and microanalysis data. More information about the file format and its specifications can be found here.

The library is designed to be minimalist, leaving post-processing of the data to the user’s script. The only dependency of pyHMSA is to NumPy, in order to represent the multi-dimensional data.

pyHMSA is written to support both Python 2 and 3.

The library is provided under the MIT License.

pyHMSA was developed as part of the doctorate thesis project of Philippe T. Pinard at RWTH Aachen University (Aachen, Germany) under the supervision of Dr. Silvia Richter.
A few examples from the pyHMSA library.

1.1 Create a new data file

This example shows how to create a new data file object from scratch and add one condition and one data set.

1.1.1 Create main object

First, we import the `DataFile` class and define a `datafile` object:

```python
from pyhmsa.datafile import DataFile
datafile = DataFile()
```

From the `datafile` object, the header can be modified using the attribute `header`, conditions can be added/modified/removed using the `conditions` which acts as a Python’s dictionary and data sets can be added/modified/removed in the same way as conditions using the `data`.

1.1.2 Header

Let’s personalize the data file and specify values from some of the default header fields. The default header fields are: `title`, `author`, `owner`, `date`, `time`, `timezone` and `checksum`. The checksum field is automatically determined when saving a data file. The default header fields can be set using their respective attributes or using lowercase keys. The instance setting the title as follows:

```python
datafile.header.title = 'Example'
```

is equivalent to this:

```python
datafile.header['title'] = 'Example'
```

The `date` field of the header is stored as a `datetime.date` object as specified in Python’s standard library. The same goes for the `time`, stored as a `datetime.time` object. For more information on these objects, refer to Python’s standard library. For a new data file, the date and time will often correspond to the current date and time. A shortcut to set both is to use Python’s `datetime.datetime` object.

```python
import datetime
datafile.header.datetime = datetime.datetime.now()
```
1.1.3 Condition

The next step is to create a condition. For this example, we will create an acquisition point condition (`AcquisitionPoint`). This condition requires to define a position. Positions are specified by the `SpecimenPosition` class. Note that per the HMSA specification, the specimen position could also be a condition on its own. All arguments of the specimen position are optional. We will define \( x \), \( y \) and \( z \).

```python
from pyhmsa.spec.condition.specimenposition import SpecimenPosition
position = SpecimenPosition(x=0.0, y=1.0, z=2.0)
```

With this position we can create our acquisition condition object. The optional arguments (`dwell_time`, `total_time`, `dwell_time_live`) can be defined later.

```python
from pyhmsa.spec.condition.acquisition import AcquisitionPoint
acq = AcquisitionPoint(position=position)
```

Numerical attributes with a magnitude (i.e. unit) can be defined in three different ways:

- Using the set method and specify the unit as the second argument.
  ```python
  acq.set_dwell_time(5.0, 's')
  ```

- Using directly the attribute and a 2-item tuple where the first item is the value and the second the unit.
  ```python
  acq.dwell_time = (5.0, 's')
  ```

- Not specifying the unit. The default unit will be used. Refer to the documentation to know which unit is assigned by default.
  ```python
  acq.dwell_time = 5.0
  ```

Note that regardless how a numerical attribute is set, all get methods will return a special object called `arrayunit`. The details of this object is not important for the users, except that it behaves as a regular number (i.e. Python’s `float`) and that it has an extra attribute `unit` to retrieve the unit. For example,

```python
print(acq.dwell_time)  # Returns: 5.0 s
print(acq.get_dwell_time())  # Returns: 5.0 s
print(acq.dwell_time.unit)  # Returns: s
```

Finally, we can add our acquisition point condition to the `datafile` object. We use the identifier `Acq0`.

```python
datafile.conditions[‘Acq0’] = acq
```

1.1.4 Data set

All data set classes are derived from NumPy’s arrays. NumPy is a powerful Python library to handle multi-dimensional arrays. It also allows to define the type of data (integer, float) and the number of bytes. Based on the HMSA specifications the following NumPy data types are allowed: `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `float32` and `float64`. If you want to perform advanced array manipulation, we refer you to the NumPy’s documentation.

For this example, we will create a 1D analysis data set (`Analysis1D`). This data set is designed to store a measurement at a single point in space or time with one datum dimension. It has no collection dimensions. An example of such data set would be an EDS spectrum.

We create the data set with 1000 channels using the `int64` data type.

```python
import numpy as np
from pyhmsa.spec.datum.analysis import Analysis1D
```
channels = 1000
datum = Analysis1D(channels, np.int64)

At this point, the data set does not contain any values. No assumption can be made on the initial values. For the purpose of this example, we will fill the array using random numbers generated from 0 to 5000.

```python
import random
datum[: ] = [random.randint(0, 5000) for _ in range(channels)]
```

As for the condition, we add the new data set to the `datafile` object.

datafile.data[‘Spectrum0’] = datum

### 1.1.5 Saving

Now our data file is created, we obviously would like to save it. The `datafile` object was an easy utility method `write` which allows to save the object to disk. The method takes one argument, the location (absolute or relative) where to save the data file. Note that the extension of the file can either be `.xml` or `.hmsa`. Both files will automatically be created as per the HMSA specifications.

datafile.write(‘example1.xml’)

Once a `datafile` object has been saved, it can be saved again to the same location by just calling `write()` without any argument.

datafile.write()  # Save to the same location

### 1.1.6 Full source code

```python
#!/usr/bin/env python
from pyhmsa.datafile import DataFile
datafile = DataFile()

# Header
datafile.header.title = ‘Example’
datafile.header[‘title’] = ‘Example’
import datetime
datafile.header.datetime = datetime.datetime.now()

# Condition
## Create a position
from pyhmsa.spec.condition.specimenposition import SpecimenPosition
position = SpecimenPosition(x=0.0, y=1.0, z=2.0)

## Create an acquisition condition
from pyhmsa.spec.condition.acquisition import AcquisitionPoint
acq = AcquisitionPoint(position=position)
acq.set_dwell_time(5.0, ‘s’)  # Returns: 5.0 s
acq.dwell_time = (5.0, ‘s’)  # Returns: 5.0 s
acq.dwell_time = 5.0  # Returns: s
```

1.1. Create a new data file
## Add condition to datafile with the ID = Acq0

datafile.conditions['Acq0'] = acq

# Dataset
## Create a dataset (NumPy array) of 1000 64-bit integers

import numpy as np
from pyhmsa.spec.datum.analysis import Analysis1D
channels = 1000
datum = Analysis1D(channels, np.int64)

## Assign values to the array by generating a random integer between 0 and 5000

import random
datum[:] = [random.randint(0, 5000) for _ in range(channels)]

## Add dataset to datafile with the ID = Spectrum0

datafile.data['Spectrum0'] = datum

# Save datafile to a file
## Only one of the two files need to be specified, either .xml or .hmsa

datafile.write('example1.xml')
datafile.write()  # Save to the same location

### 1.2 Read a data file

In this example, we will read one of the example HMSA data files provided by the authors: Brecci EDS spectrum. You can download the data file here.

We first import the `DataFile` class.

```python
from pyhmsa.datafile import DataFile
```

Then assuming that the Breccia EDS spectrum files are in the same folder as this script, we simply called the class method `read`.

```python
datafile = DataFile.read('Breccia - EDS sum spectrum.xml')
```

And that’s all!

#### 1.2.1 Advanced

Reading a large data file may take a long time since all the information is transferred in the memory. To get a progress report or to prevent blocking operation, the pyHMSA API provides an advanced reader `DataFileReader` which operates inside a thread. It can be used as follows:

```python
import time
from pyhmsa.fileformat.datafile import DataFileReader
reader = DataFileReader()
reader.read('Breccia - EDS sum spectrum.xml')

while reader.is_alive():
    print('{0:.n}% - {1}'.format(reader.progress * 100.0, reader.status))
    time.sleep(0.1)
print('{0:.n}% - {1}'.format(reader.progress * 100.0, reader.status))
```
Note that the read only initiates the reading process, but does not any data file. The method get must be called to return the data file.

### 1.2.2 Full source code

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

from pyhmsa.datafile import DataFile
datafile = DataFile.read('Breccia - EDS sum spectrum.xml')

# Advanced
import time
from pyhmsa.fileformat.datafile import DataFileReader
reader = DataFileReader()
reader.read('Breccia - EDS sum spectrum.xml')

while reader.is_alive:
    print('{0:n}% - {1}'.format(reader.progress * 100.0, reader.status))
    time.sleep(0.1)
    print('{0:n}% - {1}'.format(reader.progress * 100.0, reader.status))
datafile = reader.get()
```

### 1.3 Assign a condition to a data set

In the first example, we create an acquisition point condition, but only added it to the conditions inside the data file object. Conditions can also be, and in many cases, should be associated directly with the data set(s) they are representing. It is often the case that the same condition is shared between several data sets. For instance, the condition describing the instrument (Instrument) is usually applicable to all collected data sets. So, how to add conditions to data sets?

Let’s assume we have a data set named `datum` and a condition named `acq` (as in the first example):

```python
# Condition
from pyhmsa.spec.condition.specimenposition import SpecimenPosition
from pyhmsa.spec.condition.acquisition import AcquisitionPoint
position = SpecimenPosition(x=0.0, y=1.0, z=2.0)
acq = AcquisitionPoint(position=position)
acq.set_dwell_time(5.0, 's')

# Dataset
import random
import numpy as np
from pyhmsa.spec.datum.analysis import Analysis1D
channels = 1000
datum = Analysis1D(channels, np.int64)
datum[:] = [random.randint(0, 5000) for _ in range(channels)]
```

In the current state, the data set nor the condition are added to the data file object. Each data set object has a conditions attribute that acts exactly as the conditions attribute of the `DataFile` object. We can then simply add the condition as we did before.

```python
datum.conditions['Acq0'] = acq
```
Now let's add the datum to the data file object.

datafile.data[‘Spectrum’] = datum

That's all, but there is some magic that also happened when adding the data set to the data file object. If we list the data file's conditions, we will see that our condition, added to the data set, is also present.

```python
print(list(datafile.conditions))  # Returns: [‘Acq0’]
```

The `conditions` attribute of the data file object contains all conditions of the whole data file object. We can technically retrieve and modify the same condition from two locations: the data file or the data set.

```python
assert datafile.conditions[‘Acq0’] is datafile.data[‘Spectrum’].conditions[‘Acq0’]
```

Note that the order of the operation is not important. The conditions added to a data set will always be added to the overall conditions of the data file.

Some precisions must be made regarding removing conditions. If a condition is removed from the data file, it will also be removed from all data sets having this condition.

```python
del datafile.conditions[‘Acq0’]
print(list(datafile.conditions))  # Returns: []
print(list(datafile.data[‘Spectrum’].conditions))  # Returns: []
```

However, if a condition is removed from a data set, it will only be removed from this data set and not from the data file.

```python
datafile.data[‘Spectrum’].conditions[‘Acq0’] = acq  # Reset
del datafile.data[‘Spectrum’].conditions[‘Acq0’]
print(list(datafile.conditions))  # Returns: [‘Acq0’]
print(list(datafile.data[‘Spectrum’].conditions))  # Returns: []
```

This behavior may appear counter-intuitive but it is not possible to know if that condition was added first to a data set or directly to the data file’s conditions.

### 1.3.1 Full source code

```python
#!/usr/bin/env python
rom pyhmsa.datafile import DataFile

datafile = DataFile()

# Condition
from pyhmsa.spec.condition.specimenposition import SpecimenPosition
from pyhmsa.spec.condition.acquisition import AcquisitionPoint
position = SpecimenPosition(x=0.0, y=1.0, z=2.0)
acq = AcquisitionPoint(position=position)
acq.set_dwell_time(5.0, ‘s’)

# Dataset
import random
import numpy as np
from pyhmsa.spec.datum.analysis import Analysis1D
channels = 1000
datum = Analysis1D(channels, np.int64)
datum[:]= [random.randint(0, 5000) for _ in range(channels)]

# Assign condition
```
datum.conditions['Acq0'] = acq

# Add dataset
datafile.data['Spectrum'] = datum

# Check
print(list(datafile.conditions))  # Returns: ['Acq0']
assert datafile.conditions['Acq0'] is datafile.data['Spectrum'].conditions['Acq0']

# Removing globally
del datafile.conditions['Acq0']
print(list(datafile.conditions))  # Returns: []
print(list(datafile.data['Spectrum'].conditions))  # Returns: []

# Removing locally
datafile.data['Spectrum'].conditions['Acq0'] = acq  # Reset
del datafile.data['Spectrum'].conditions['Acq0']
print(list(datafile.conditions))  # Returns: ['Acq0']
print(list(datafile.data['Spectrum'].conditions))  # Returns: []

1.4 Extract a spectrum to a CSV file

We show in this example how to extract a spectrum from a HMSA data file and save it in a CSV file with Python. As in the second example, we use one of the example HMSA data files provided by the authors: Brecci EDS spectrum. You can download the data file here.

First, let’s read the data file.

```python
from pyhmsa.datafile import DataFile
datafile = DataFile.read('Breccia - EDS sum spectrum.xml')
```

Then, we must find the data set corresponding to our spectrum. In this case, the data set is called EDS sum spectrum so we could retrieve it from its ID.

```python
spectrum = datafile.data['EDS sum spectrum']
```

However, in some cases, we might not know the name of a data set or an only a portion of the name. The library offers search methods to find both data sets and conditions.

For instance, we could search for all data sets containing the word spectrum as follow:

```python
results = datafile.data.findvalues('*spectrum*')
print(len(results))  # Returns: 1
```

The * in the search pattern are wild cards and indicates to match any character.

We could also search based on the type of data set. In this case, we are looking for an Analysis1D data set.

```python
from pyhmsa.spec.datum.analysis import Analysis1D
results = datafile.data.findvalues(Analysis1D)
print(len(results))  # Returns: 1
```

Once we have our data set, we can use the utility method get_xy to retrieve a two-dimensional array where the first column contains x values and the second y values. This method is particularly useful since it will search through the associated conditions to the data set to see if a calibration was defined for the x values. In this example, a linear calibration with an offset of -237.098251 was defined. The first x value should therefore be equal to this value.
```python
spectrum = next(iter(results))  # Take first result
xy = spectrum.get_xy()
print(xy[0, 0])  # Returns -237.098251

The get_xy can also returns labels for the x and y values as defined in the conditions.
xlabel, ylabel, xy = spectrum.get_xy(with_labels=True)

Finally, we can use Python’s csv module to create the CSV file.

```import csv
with open(‘breccia.csv’, ‘w’) as fp:
    writer = csv.writer(fp)  # Create CSV writer
    writer.writerow([xlabel, ylabel])  # Header
    writer.writerows(xy)
```

### 1.4.1 Full source code

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

from pyhmsa.datafile import DataFile
datafile = DataFile.read(‘Breccia - EDS sum spectrum.xml’)
spectrum = datafile.data[‘EDS sum spectrum’]

# Search
results = datafile.data.findvalues(‘*spectrum*’)
print(len(results))  # Returns: 1

from pyhmsa.spec.datum.analysis import Analysis1D
results = datafile.data.findvalues(Analysis1D)
print(len(results))  # Returns: 1

spectrum = next(iter(results))  # Take first result
xy = spectrum.get_xy()
print(xy[0, 0])  # Returns -237.098251

xlabel, ylabel, xy = spectrum.get_xy(with_labels=True)

# Save
import csv
with open(‘breccia.csv’, ‘w’) as fp:
    writer = csv.writer(fp)  # Create CSV writer
    writer.writerow([xlabel, ylabel])  # Header
    writer.writerows(xy)
```

### 1.5 Use alternative language

Although ASCII characters are preferred throughout the HMSA specifications, it is possible to provide information in Unicode character and alternative spelling.

In pyHMSA, this is done through a special object type called langstr. This object behaves exactly like the default Python’s str type with the exception that alternative spelling can be provided. Let’s look at an example how to specify an author’s name in two different languages.
First we create a data file object and import the `langstr` type.

```python
from pyhmsa.datafile import DataFile
from pyhmsa.type.language import langstr
datafile = DataFile()
```

Then we create a new `langstr` object for the author’s name. The first argument of `langstr` is the name in English (i.e. ASCII characters). The second argument is a dictionary where the key is a valid language code and/or country code, as specified by ISO 639-1 and ISO 3166, respectively.

```python
author = langstr('Wilhelm Conrad Roentgen', {'de': u'Wilhelm Conrad Röntgen'})
datafile.header.author = author
```

The alternative spellings of a string can be access using the attribute `alternatives` which returns a dictionary. Note that once created a `langstr` object is immutable; it cannot be modified.

```python
print(datafile.header.author.alternatives['de'])  # Returns ...
```

### 1.5.1 Full source code

```python
#!/usr/bin/env python
from pyhmsa.datafile import DataFile
from pyhmsa.type.language import langstr
datafile = DataFile()

author = langstr('Wilhelm Conrad Roentgen’, {'de’: u’Wilhelm Conrad Röntgen’})
datafile.header.author = author

print(datafile.header.author.alternatives['de'])  # Returns ...
```

### 1.6 Create an Elemental ID x-ray condition

Specifying the x-ray line in an `ElementalIDXray` condition is slightly different than other attributes, so this example shows how to do it. In microanalysis, there exists two common types of nomenclature for x-ray line: the one proposed by the IUPAC (e.g. \( K-L_3 \)) and the traditional Siegbahn notation (\( K\alpha_1 \)). HMSA specifications does not enforce one notation over the other and encourage the users to specify the x-ray line in both notations.

In a similar way as for alternative languages (see the example on *Use alternative language*), a new type is defined for x-ray lines: `xrayline`. The type takes two required arguments (the x-ray line and its notation) and an optional argument for the x-ray line express in the other notation. For example, we can specify the \( K\alpha_1 \) line as follows:

```python
from pyhmsa.type.xrayline import xrayline, NOTATION_SIEGBAHN
line = xrayline('Ka1', NOTATION_SIEGBAHN, 'K-L3')
```

The alternative value is automatically interpreted to be expressed in the IUPAC notation.

This new `line` object can then be used to create a new `ElementalIDXray` condition.

```python
from pyhmsa.spec.condition.elementalid import ElementalIDXray
condition = ElementalIDXray(29, line, (8047.82, 'eV'))
print(condition)  # Returns: <ElementalIDXray(atomic_number=29, energy=8047.82 eV, line=Ka1)>
```

1.6. Create an Elemental ID x-ray condition 11
1.6.1 Full source code

#!/usr/bin/env python

from pyhmsa.type.xrayline import xrayline, NOTATION_SIEGBAHN
line = xrayline('Ka1', NOTATION_SIEGBAHN, 'K-L3')

from pyhmsa.spec.condition.elementalid import ElementalIDXray
condition = ElementalIDXray(29, line, (8047.82, 'eV'))
print(condition)  # Returns: <ElementalIDXray(atomic_number=29, energy=8047.82 eV, line=Ka1)>
The API follows closely the name convention, hierarchy and parameter names of the HMSA specifications. The type of conditions and datasets available can be found below.

2.1 Header

class pyhmsa.spec.header.Header

    author
    author

    checksum
    checksum

    date
    date

    owner
    legal owner

    time
    time

    timezone
    timezone

    title
    title

2.2 Conditions

List of available conditions:

2.2.1 Acquisition

Conditions containing the parameters used for the acquisition of the data.
**Constants**

**Raster modes**

```python
pyhmsa.spec.condition.acquisition.RASTER_MODE_STAGE
pyhmsa.spec.condition.acquisition.RASTER_MODE_BEAM
```

**Z raster modes**

```python
pyhmsa.spec.condition.acquisition.RASTER_MODE_Z_FIB
```

**Position locations**

```python
pyhmsa.spec.condition.acquisition.POSITION_LOCATION_START
pyhmsa.spec.condition.acquisition.POSITION_LOCATION_CENTER
pyhmsa.spec.condition.acquisition.POSITION_LOCATION_END
```

**Classes**

```python
class pyhmsa.spec.condition.acquisition.AcquisitionPoint (position, dwell_time=None, total_time=None, dwell_time_live=None)
```

Defines the position and duration for a singular measurement of the specimen.

**Parameters**

- **position** – physical location on (or in) the specimen (required)
- **dwell_time** – uniform real time taken for each individual measurement (optional)
- **total_time** – total real time taken to collect all measurements (optional)
- **dwell_time_live** – analogous detector live time for each individual measurement (optional)

**CLASS** = ‘Point’

**TEMPLATE** = ‘Acquisition’

- **dwell_time**
  uniform real time taken for each individual measurement

- **dwell_time_live**
  analogous detector live time for each individual measurement

- **get_dwell_time**
  ```python
  get_dwell_time(instance)
  ```

- **get_dwell_time_live**
  ```python
  get_dwell_time_live(instance)
  ```

- **get_position**
  ```python
  get_position(instance)
  ```

- **get_total_time**
  ```python
  get_total_time(instance)
  ```

- **position**
  physical location on (or in) the specimen

- **set_dwell_time**
  ```python
  set_dwell_time(instance, value, unit=None)
  ```

- **set_dwell_time_live**
  ```python
  set_dwell_time_live(instance, value, unit=None)
  ```
**set_position** *(instance, value)*

**set_total_time** *(instance, value, unit=None)*

**total_time**

  total real time taken to collect all measurements

```python
class pyhmsa.spec.condition.acquisition.AcquisitionMultipoint (positions=None, dwell_time=None, total_time=None, dwell_time_live=None)
```

Defines the position and duration of an irregular sequence of measurements of the specimen.

**Parameters**

- **positions** – iterable of specimen positions (optional)
- **dwell_time** – uniform real time taken for each individual measurement (optional)
- **total_time** – total real time taken to collect all measurements (optional)
- **dwell_time_live** – analogous detector live time for each individual measurement (optional)

**CLASS** = ‘Multipoint’

**TEMPLATE** = ‘Acquisition’

**dwell_time**

  uniform real time taken for each individual measurement

**dwell_time_live**

  analogous detector live time for each individual measurement

```python
get_dwell_time (instance)
get_dwell_time_live (instance)
get_positions (instance)
get_total_time (instance)
```

**positions**

  specimen positions

```python
set_dwell_time (instance, value, unit=None)
set_dwell_time_live (instance, value, unit=None)
set_total_time (instance, value, unit=None)
```

**total_time**

  total real time taken to collect all measurements

```python
class pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan (step_count, step_size=None, frame_count=None, position_start=None, position_end=None, raster_mode=None, dwell_time=None, total_time=None, dwell_time_live=None)
```
Defines the position and duration of a one-dimensional raster over the specimen. Applies only to a linear sequence of steps, using equal step sizes and dwell times for each measurement. For irregular step sizes, refer to `AcquisitionMultiPoint`.

**Parameters**

- `step_count` – number of steps (required)
- `step_size` – dimension of each step (optional)
- `position_start` – start position (optional)
- `position_end` – end position (optional)
- `raster_mode` – mode of rastering, `RASTER_MODE_STAGE` or `RASTER_MODE_BEAM` (optional)
- `dwell_time` – uniform real time taken for each individual measurement (optional)
- `total_time` – total real time taken to collect all measurements (optional)
- `dwell_time_live` – analogous detector live time for each individual measurement (optional)

**CLASS** = ‘Raster/LineScan’

**TEMPLATE** = ‘Acquisition’

- `dwell_time`
  uniform real time taken for each individual measurement

- `dwell_time_live`
  analogous detector live time for each individual measurement

- `frame_count`
  number of accumulated frames

- `get_dwell_time(instance)`

- `get_dwell_time_live(instance)`

- `get_frame_count(instance)`

- `get_position_end()`
  Returns the end position.

  Returns end position

  Return type `SpecimenPosition`

- `get_position_start()`
  Returns the start position.

  Returns start position

  Return type `SpecimenPosition`

- `get_positions(instance)`

- `get_raster_mode(instance)`

- `get_step_count(instance)`

- `get_step_size(instance)`

- `get_total_time(instance)`

- `position_end`
  End position
position_start
Start position

positions
defined physical location(s) of the raster

raster_mode
mode of rastering

set_dwell_time(instance, value, unit=None)
set_dwell_time_live(instance, value, unit=None)
set_frame_count(instance, value, unit=None)
set_position_end(value)
Sets the end position.

Parameters value (SpecimenPosition) – end position

set_position_start(value)
Sets the start position.

Parameters value (SpecimenPosition) – start position

set_raster_mode(instance, value)
set_step_count(instance, value, unit=None)
set_step_size(instance, value, unit=None)
set_total_time(instance, value, unit=None)

step_count
number of steps

step_size
dimension of each step

total_time
total real time taken to collect all measurements

class pyhmsa.spec.condition.acquisition.AcquisitionRasterXY(step_count_x,
step_count_y,
step_size_x=None,
step_size_y=None,
frame_count=None,
position=None,
raster_mode=None,
dwell_time=None,
total_time=None,
dwell_time_live=None)

Defines the position and duration of a two-dimensional X/Y raster over the specimen.

Parameters

• step_count_x – number of steps in x direction (required)
• step_count_y – number of steps in y direction (required)
• step_size_x – dimension of each step in x direction (optional)
• step_size_y – dimension of each step in y direction (optional)
• frame_count – number of accumulated frames (optional)
- **position** – specimen position (optional)
- **raster_mode** – mode of rastering, `RASTER_MODE_STAGE` or `RASTER_MODE_BEAM` (optional)
- **dwell_time** – uniform real time taken for each individual measurement (optional)
- **total_time** – total real time taken to collect all measurements (optional)
- **dwell_time_live** – analogous detector live time for each individual measurement (optional)

```
CLASS = 'Raster/XY'
TEMPLATE = 'Acquisition'
```

dwell_time
uniform real time taken for each individual measurement
dwell_time_live
analogous detector live time for each individual measurement

```
frame_count
number of accumulated frames
```

get_dwell_time(instance)
get_dwell_time_live(instance)
get_frame_count(instance)

```
get_position(include_location=False)
Returns the physical location on (or in) the specimen.

    Returns  specimen position
    Return type  SpecimenPosition
```

get_positions(instance)
get_raster_mode(instance)
get_step_count_x(instance)
get_step_count_y(instance)
get_step_size_x(instance)
get_step_size_y(instance)
get_total_time(instance)

```
position
Physical location on (or in) the specimen
```

```
positions
defined physical location(s) of the raster
```

```
raster_mode
mode of rastering
```

```
set_dwell_time(instance, value, unit=None)
set_dwell_time_live(instance, value, unit=None)
set_frame_count(instance, value, unit=None)
set_position(value, loc=None)
```
Sets the physical location on (or in) the specimen.
Parameters

- **value** *(SpecimenPosition)* – specimen position
- **loc** – location, either `POSITION_LOCATION_START` or `POSITION_LOCATION_CENTER`

```python
def set_raster_mode(self, instance, value):
    pass
def set_step_count_x(self, instance, value, unit=None):
    pass
def set_step_count_y(self, instance, value, unit=None):
    pass
def set_step_size_x(self, instance, value, unit=None):
    pass
def set_step_size_y(self, instance, value, unit=None):
    pass
def set_total_time(self, instance, value, unit=None):
    pass
```

- **step_count_x**
  number of steps in x direction
- **step_count_y**
  number of steps in y direction
- **step_size_x**
  dimension of each step in x direction
- **step_size_y**
  dimension of each step in y direction
- **total_time**
  total real time taken to collect all measurements

```python
class AcquisitionRasterXYZ:
    def __init__(self, step_count_x=None, step_count_y=None, step_count_z=None, step_size_x=None, step_size_y=None, step_size_z=None, position=None, raster_mode_z=None, raster_mode=None, dwell_time=None, total_time=None, dwell_time_live=None):
        pass
```

Defines the position and duration of a three-dimensional X/Y/Z raster over the specimen.

Parameters

- **step_count_x** – number of steps in x direction (required)
- **step_count_y** – number of steps in y direction (required)
- **step_count_z** – number of steps in z direction (required)
- **step_size_x** – dimension of each step in x direction (optional)
- **step_size_y** – dimension of each step in y direction (optional)
- **step_size_z** – dimension of each step in z direction (optional)
- **position** – specimen position (optional)
- **raster_mode_z** – mode of rastering in z direction, `RASTER_MODE_Z_FIB` (optional)
• **raster_mode** – mode of rastering, `RASTER_MODE_STAGE` or `RASTER_MODE_BEAM` (optional)
• **dwell_time** – uniform real time taken for each individual measurement (optional)
• **total_time** – total real time taken to collect all measurements (optional)
• **dwell_time_live** – analogous detector live time for each individual measurement (optional)

```python
CLASS = 'Raster/XYZ'
TEMPLATE = 'Acquisition'
dwell_time
uniform real time taken for each individual measurement
dwell_time_live
analogous detector live time for each individual measurement
get_dwell_time (instance)
get_dwell_time_live (instance)
get_position (include_location=False)
    Returns the physical location on (or in) the specimen.
        Returns specimen position
        Return type SpecimenPosition
get_positions (instance)
get_raster_mode (instance)
get_raster_mode_z (instance)
get_step_count_x (instance)
get_step_count_y (instance)
get_step_count_z (instance)
get_step_size_x (instance)
get_step_size_y (instance)
get_step_size_z (instance)
get_total_time (instance)
```

**position**
Physical location on (or in) the specimen

**positions**
defined physical location(s) of the raster

**raster_mode**
mode of rastering

**raster_mode_z**
mode of rastering in z direction

```python
set_dwell_time (instance, value, unit=None)
set_dwell_time_live (instance, value, unit=None)
set_position (value, loc=None)
    Sets the physical location on (or in) the specimen.
```
Parameters

- **value** *(SpecimenPosition)* – specimen position
- **loc** – location, either *POSITION_LOCATION_START* or *POSITION_LOCATION_CENTER*

```python
set_raster_mode (instance, value)
set_raster_mode_z (instance, value)
set_step_count_x (instance, value, unit=None)
set_step_count_y (instance, value, unit=None)
set_step_count_z (instance, value, unit=None)
set_step_size_x (instance, value, unit=None)
set_step_size_y (instance, value, unit=None)
set_step_size_z (instance, value, unit=None)
set_total_time (instance, value, unit=None)
```

```python
step_count_x
    number of steps in x direction
step_count_y
    number of steps in y direction
step_count_z
    number of steps in z direction
step_size_x
    dimension of each step in x direction
step_size_y
    dimension of each step in y direction
step_size_z
    dimension of each step in z direction
```

total_time
    total real time taken to collect all measurements

### 2.2.2 Composition

Conditions describing the composition of a material.

Classes

```python
class pyhmsa.spec.condition.composition.CompositionElemental (unit, values=None, **kwargs)
```

Defines the composition of a material in terms of its constituent elements. The composition is a *dict* where the keys are atomic numbers and the values the amounts of an element.

**Parameters**

- **unit** – unit in which the composition is defined (required)
- **values**
- ****kwargs

**CLASS** = ‘Elemental’

**TEMPLATE** = ‘Composition’

clear () → None. Remove all items from D.
get \((k, d)\) \(\rightarrow\) \(D[k]\) if \(k\) in \(D\), else \(d\). \(d\) defaults to None.

get_unit \((\text{instance})\)

items () \(\rightarrow\) list of \(D\)'s (key, value) pairs, as 2-tuples

iteritems () \(\rightarrow\) an iterator over the (key, value) items of \(D\)

iterkeys () \(\rightarrow\) an iterator over the keys of \(D\)

itervalues () \(\rightarrow\) an iterator over the values of \(D\)

keys () \(\rightarrow\) list of \(D\)'s keys

pop \((k, d)\) \(\rightarrow\) \(v\), remove specified key and return the corresponding value.

If key is not found, \(d\) is returned if given, otherwise \(\text{KeyError}\) is raised.

popitem () \(\rightarrow\) \((k, v)\), remove and return some (key, value) pair as a 2-tuple; but raise \(\text{KeyError}\) if \(D\) is empty.

set_unit \((\text{instance, value})\)

setdefault \((k, d)\) \(\rightarrow\) \(D.get(k,d)\), also set \(D[k]=d\) if \(k\) not in \(D\)

to_wt ()

Returns a CompositionElemental with unit of \(\text{wt\%}\).

unit

unit in which the composition is defined

update \((E, **F)\) \(\rightarrow\) None. Update \(D\) from mapping/iterable \(E\) and \(F\).

If \(E\) present and has a .keys() method, does: for \(k\) in \(E\): \(D[k]=E[k]\)

If \(E\) present and lacks .keys() method, does: for \((k, v)\) in \(E\): \(D[k]=v\)

In either case, this is followed by: for \(k, v\) in \(F.items()\): \(D[k]=v\)

values () \(\rightarrow\) list of \(D\)'s values

### 2.2.3 Detector

Conditions describing the type and configuration of a detector used to collect the data.

**Constants**

**Signal types**

pyhmsa.spec.condition.detector.SIGNAL_TYPE_EDS

pyhmsa.spec.condition.detector.SIGNAL_TYPE_WDS

pyhmsa.spec.condition.detector.SIGNAL_TYPE_ELS

pyhmsa.spec.condition.detector.SIGNAL_TYPE_AES

pyhmsa.spec.condition.detector.SIGNAL_TYPE_PES

pyhmsa.spec.condition.detector.SIGNAL_TYPE_XRF

pyhmsa.spec.condition.detector.SIGNAL_TYPE_CLS

pyhmsa.spec.condition.detector.SIGNAL_TYPE_GAM

pyhmsa.spec.condition.detector.SIGNAL_TYPE_EBSD

pyhmsa.spec.condition.detector.SIGNAL_TYPE_BEI
pyhmsa.spec.condition.detector.SIGNAL_TYPE_SEI

Collection modes

pyhmsa.spec.condition.detector.COLLECTION_MODE_PARALLEL
pyhmsa.spec.condition.detector.COLLECTION_MODE_SERIAL

PHA modes (WDS signal)

pyhmsa.spec.condition.detector.PHA_MODE_INTEGRAL
pyhmsa.spec.condition.detector.PHA_MODE_DIFFERENTIAL

XEDS technologies (EDS signal)

pyhmsa.spec.condition.detector.XEDS_TECHNOLOGY_GE
pyhmsa.spec.condition.detector.XEDS_TECHNOLOGY_SILI
pyhmsa.spec.condition.detector.XEDS_TECHNOLOGY_SDD
pyhmsa.spec.condition.detector.XEDS_TECHNOLOGY_UCAL

Helper classes

Calibration

class pyhmsa.spec.condition.calibration.CalibrationConstant(quantity, unit, value)

Defines the energy/wavelength/etc calibration of a spectrometer or other measurement device operating at a fixed position, such as a CL monochromator.

Parameters

- quantity – physical quantity (required)
- unit – unit (required)
- value – value (required)

get_index(value)

get_quantity(index)

get_unit(instance)

get_value(instance)

quantity

physical quantity

set_quantity(instance, value)

set_unit(instance, value)

set_value(instance, value, unit=None)

unit

unit
value
  constant value

class pyhmsa.spec.condition.calibration.CalibrationLinear(quantity, unit, gain, offset)
  Defines the energy/wavelength/etc calibration of a spectrometer or other measurement device, for which the measurement ordinals (e.g. channel numbers) have a linear relationship to the physical quantity (e.g. nm), with a constant offset and gain.

  Parameters
  • quantity – physical quantity (required)
  • unit – unit (required)
  • gain – gain (required)
  • offset – offset, the calibration value (energy, wavelength, position, etc.) corresponding to the first measurement ordinal (required)

  .. method:: gain
    :self:
    :returns: gain
  .. method:: get_gain(instance)
  .. method:: get_index(value)
  .. method:: get_offset(instance)
  .. method:: get_quantity(index)
  .. method:: get_unit(instance)
  .. method:: offset
    :self:
    :returns: offset
  .. method:: quantity
    :self:
    :returns: physical quantity
  .. method:: set_gain(instance, value, unit=None)
  .. method:: set_offset(instance, value, unit=None)
  .. method:: set_quantity(instance, value)
  .. method:: set_unit(instance, value)

class pyhmsa.spec.condition.calibration.CalibrationPolynomial(quantity, unit, coefficients)
  Defines the energy/wavelength/etc calibration of a spectrometer or other measurement device, for which the measurement ordinals (e.g. channel numbers) have a relationship to the physical quantity (e.g. nm) that may be modelled by an nth order polynomial.

  Parameters
  • quantity – physical quantity (required)
  • unit – unit (required)
  • coefficients – iterable of coefficients (required)

coefficients
  polynomial coefficients
class pyhmsa.spec.condition.calibration.CalibrationExplicit(quantity, unit, values, labels=None)

Defines the energy/wavelength/etc calibration of a spectrometer or other measurement device, for which relationship between the measurement ordinals (e.g. channel numbers) and physical quantity (e.g. nm) cannot be adequately modelled by linear or polynomial functions, and therefore must be declared explicitly for each ordinal as an array of floating point values.

Parameters

- **quantity** – physical quantity (required)
- **unit** – unit (required)
- **values** – explicit values (required)

get_index(value)

get_label(index)

get_labels(instance)

get_quantity(index)

get_unit(instance)

get_values(instance)

labels
text labels for each of the calibration points

quantity
physical quantity

set_labels(instance, value)

set_quantity(instance, value)

set_unit(instance, value)

set_values(instance, value, unit=None)

unit

values
explicit values
Window

class pyhmsa.spec.condition.detector.WindowLayer(material, thickness)
   Defines a layer of a window.
   Parameters
   • material – material
   • thickness – thickness

get_material(instance)
get_thickness(instance)
material
   material
set_material(instance, value)
set_thickness(instance, value, unit=None)
thickness
   thickness

class pyhmsa.spec.condition.detector.Window(layers=None)
   Defines the layer(s) of a window.
   Parameters layers – iterable of Layer (optional)

append_layer(material, thickness)
   Helper function that creates a new Layer and appends it to this window.
   Parameters
   • material – material
   • thickness – thickness
   Returns created layer
   Return type Layer

get_layers(instance)
layers
   modifiable list of layers

PHA

class pyhmsa.spec.condition.detector.PulseHeightAnalyser(bias=None, gain=None, base_level=None, window=None, mode=None)

   Defines the condition of the pulse height analyser of a WDS spectrometer.
   Parameters
   • bias – bias (optional)
   • gain – gain (optional)
   • base_level – base level (optional)
   • window – window (optional)
mode – mode, either PHA_MODE_INTEGRAL or PHA_MODE_DIFFERENTIAL (optional)

base_level
base level
bias
bias
gain
gain
get_base_level(instance)
get_bias(instance)
get_gain(instance)
get_mode(instance)
get_window(instance)
mode
mode
set_base_level(instance, value, unit=None)
set_bias(instance, value, unit=None)
set_gain(instance, value, unit=None)
set_mode(instance)
set_window(instance, value, unit=None)
window
window

Classes

class pyhmsa.spec.condition.detector.DetectorCamera(pixel_count_u, pixel_count_v, exposure_time=None, magnification=None, focal_length=None, signal_type=None, manufacturer=None, model=None, serial_number=None, measurement_unit='counts', elevation=None, azimuth=None, distance=None, area=None, solid_angle=None, semi_angle=None, temperature=None)

Describes the calibration and collection mode of a camera used to collect a HMSA dataset, such as an EBSD or TEM camera. The camera detector is expected to have two datum axes (U and V) which are, in general, assumed to be independent of the specimen coordinate dimensions (X/Y/Z).

Parameters

• pixel_count_u – number of pixels along the horizontal axis (required)
• pixel_count_y – number of pixels along the vertical axis (required)
• exposure_time – exposure time (optional)
- `magnification` – magnification (optional)
- `focal_length` – focal length (optional)
- `signal_type` – type of signal (optional)
- `manufacturer` – manufacturer (optional)
- `model` – model (optional)
- `serial_number` – serial number (optional)
- `measurement_unit` – measurement unit (optional)
- `elevation` – elevation (optional)
- `azimuth` – azimuth (optional)
- `distance` – distance (optional)
- `area` – area (optional)
- `solid_angle` – solid angle (optional)
- `semi_angle` – semi-angle (optional)
- `temperature` – temperature (optional)

CLASS = `Camera`

TEMPLATE = `Detector`

`area`
  area

`azimuth`
  azimuth

`distance`
  distance

`elevation`
  elevation

`exposure_time`
  exposure time

`focal_length`
  focal length

`get_area` (`instance`)

`get_azimuth` (`instance`)

`get_distance` (`instance`)

`get_elevation` (`instance`)

`get_exposure_time` (`instance`)

`get_focal_length` (`instance`)

`get_magnification` (`instance`)

`get_manufacturer` (`instance`)

`get_measurement_unit` (`instance`)

`get_model` (`instance`)
get_pixel_count_u (instance)
get_pixel_count_v (instance)
get_semi_angle (instance)
get_serial_number (instance)
get_signal_type (instance)
get_solid_angle (instance)
get_temperature (instance)
magnification
manufacturer
measurement_unit
model
pixel_count_u
  number of pixels along the horizontal axis
pixel_count_v
  number of pixels along the vertical axis
semi_angle
  semi-angle
serial_number
  serial number
set_area (instance, value, unit=None)
set_azimuth (instance, value, unit=None)
set_distance (instance, value, unit=None)
set_elevation (instance, value, unit=None)
set_exposure_time (instance, value, unit=None)
set_focal_length (instance, value, unit=None)
set_magnification (instance, value, unit=None)
set_manufacturer (instance, value)
set_measurement_unit (instance, value)
set_model (instance, value)
set_pixel_count_u (instance, value, unit=None)
set_pixel_count_v (instance, value, unit=None)
set_semi_angle (instance, value, unit=None)
set_serial_number (instance, value)
set_signal_type (instance, value)
set_solid_angle (instance, value, unit=None)
set_temperature(instance, value, unit=None)

signal_type
type of signal

solid_angle
solid angle

temperature
temperature

class pyhmsa.spec.condition.detector.DetectorSpectrometer(channel_count, calibration, collection_mode=None, signal_type=None, manufacturer=None, model=None, serial_number=None, measurement_unit='counts', elevation=None, azimuth=None, distance=None, area=None, solid_angle=None, semi_angle=None, temperature=None)

Describes the calibration and collection mode of a spectrometer used to collect a HMSA dataset.

Parameters

- channel_count – number of channels (required)
- calibration (Calibration) – calibration (required)
- mode (collection) – mode of collection, either COLLECTION_MODE_PARALLEL or COLLECTION_MODE_SERIAL (optional)
- signal_type – type of signal (optional)
- manufacturer – manufacturer (optional)
- model – model (optional)
- serial_number – serial number (optional)
- measurement_unit – measurement unit (optional)
- elevation – elevation (optional)
- azimuth – azimuth (optional)
- distance – distance (optional)
- area – area (optional)
- solid_angle – solid angle (optional)
- semi_angle – semi-angle (optional)
- temperature – temperature (optional)

CLASS = 'Spectrometer'
TEMPLATE = 'Detector'

area

area
azimuth
    azimuth

    calibration
    calibration

    calibration_energy

    calibration_wavelength

    channel_count
    number of channels

    collection_mode
    mode of collection

    distance
    distance

    elevation
    elevation

    get_area (instance)

    get_azimuth (instance)

    get_calibration (instance)

    get_calibration_energy ()

    get_calibration_wavelength ()

    get_channel_count (instance)

    get_collection_mode (instance)

    get_distance (instance)

    get_elevation (instance)

    get_manufacturer (instance)

    get_measurement_unit (instance)

    get_model (instance)

    get_semi_angle (instance)

    get_serial_number (instance)

    get_signal_type (instance)

    get_solid_angle (instance)

    get_temperature (instance)

    manufacturer
    manufacturer

    measurement_unit
    measurement unit

    model
    model

    semi_angle
    semi-angle

2.2. Conditions
serial_number
serial number

set_area (instance, value, unit=None)
set_azimuth (instance, value, unit=None)
set_calibration (instance, value)
set_channel_count (instance, value, unit=None)
set_collection_mode (instance, value)
set_distance (instance, value, unit=None)
set_elevation (instance, value, unit=None)
set_manufacturer (instance, value)
set_measurement_unit (instance, value)
set_model (instance, value)
set_semi_angle (instance, value, unit=None)
set_serial_number (instance, value)
set_signal_type (instance, value)
set_solid_angle (instance, value, unit=None)
set_temperature (instance, value, unit=None)

signal_type
type of signal

solid_angle
solid angle

temperature
temperature

class pyhmsa.spec.condition.detector.DetectorSpectrometerCL (channel_count, calibration, grating_d=None, collection_mode=None, signal_type=None, manufacturer=None, model=None, serial_number=None, measurement_unit='counts', elevation=None, azimuth=None, distance=None, area=None, solid_angle=None, semi_angle=None, temperature=None)

Describes the type and configuration of a cathodoluminescence spectrometer.

Note: If the spectrometer is operating as a monochromator (e.g. monochromatic CL mapping), the calibration definition shall be of type CalibrationConstant.
Parameters

- **channel_count** – number of channels (required)
- **calibration** (Calibration) – calibration (required)
- **grating_d** – grading spacing (optional)
- **mode** (collection) – mode of collection, either COLLECTION_MODE_PARALLEL or COLLECTION_MODE_SERIAL (optional)
- **signal_type** – type of signal (optional)
- **manufacturer** – manufacturer (optional)
- **model** – model (optional)
- **serial_number** – serial number (optional)
- **measurement_unit** – measurement unit (optional)
- **elevation** – elevation (optional)
- **azimuth** – azimuth (optional)
- **distance** – distance (optional)
- **area** – area (optional)
- **solid_angle** – solid angle (optional)
- **semi_angle** – semi-angle (optional)
- **temperature** – temperature (optional)

CLASS = ‘Spectrometer/CL’

TEMPLATE = ‘Detector’

area

azimuth

calibration

calibration_energy

calibration_wavelength

channel_count

number of channels

collection_mode

mode of collection

distance

elevation

get_area (instance)

get_azimuth (instance)
get_calibration (instance)
get_calibration_energy ()
get_calibration_wavelength ()
get_channel_count (instance)
get_collection_mode (instance)
get_distance (instance)
get_elevation (instance)
get_grating_d (instance)
get_manufacturer (instance)
get_measurement_unit (instance)
get_model (instance)
get_semi_angle (instance)
get_serial_number (instance)
get_signal_type (instance)
get_solid_angle (instance)
get_temperature (instance)
grating_d
   grating spacing
manufacturer
   manufacturer
measurement_unit
   measurement unit
model
   model
semi_angle
   semi-angle
serial_number
   serial number
set_area (instance, value, unit=None)
set_azimuth (instance, value, unit=None)
set_calibration (instance, value)
set_channel_count (instance, value, unit=None)
set_collection_mode (instance, value)
set_distance (instance, value, unit=None)
set_elevation (instance, value, unit=None)
set_grating_d (instance, value, unit=None)
set_manufacturer (instance, value)
set_measurement_unit (instance, value)
set_model (instance, value)
set_semi_angle (instance, value, unit=None)
set_serial_number (instance, value)
set_signal_type (instance, value)
set_solid_angle (instance, value, unit=None)
set_temperature (instance, value, unit=None)
signal_type
type of signal
solid_angle
solid angle
temperature
temperature

class pyhmsa.spec.condition.detector.DetectorSpectrometerWDS (channel_count, calibration, collection_mode=None, dispersion_element=None, crystal_2d=None, rowland_circle_diameter=None, pulse_height_analyser=None, window=None, signal_type=None, manufacturer=None, model=None, serial_number=None, measurement_unit='counts', elevation=None, azimuth=None, distance=None, area=None, solid_angle=None, semi_angle=None, temperature=None)

Describes the type and configuration of a wavelength dispersive x-ray spectrometer.

Note: If the spectrometer is operating as a monochromator (e.g. WDS mapping), the calibration definition shall be of type CalibrationConstant.

Parameters

- channel_count – number of channels (required)
- calibration (_Calibration) – calibration (required)
- mode (collection) – mode of collection, either COLLECTION_MODE_PARALLEL or COLLECTION_MODE_SERIAL (optional)
- element (dispersion_element) – dispersion element (optional)
- `crystal_2d` – crystal 2d-spacing (optional)
- `rowland_circle_diameter` – Rowland circle diameter (optional)
- `pulse_height_analyser` (`PulseHeightAnalyser`) – pulse height analyser (optional)
- `window` (`Layer`) – window (optional)
- `signal_type` – type of signal (optional)
- `manufacturer` – manufacturer (optional)
- `model` – model (optional)
- `serial_number` – serial number (optional)
- `measurement_unit` – measurement unit (optional)
- `elevation` – elevation (optional)
- `azimuth` – azimuth (optional)
- `distance` – distance (optional)
- `area` – area (optional)
- `solid_angle` – solid angle (optional)
- `semi_angle` – semi-angle (optional)
- `temperature` – temperature (optional)

```
CLASS = 'Spectrometer/WDS'
TEMPLATE = 'Detector'
area
  area
azimuth
  azimuth
calibration
  calibration
calibration_energy
calibration_position
calibration_wavelength
channel_count
  number of channels
collection_mode
  mode of collection
crystal_2d
  crystal 2d-spacing
dispersion_element
  dispersion element
distance
  distance
elevation
  elevation
```

36 Chapter 2. API
get_area (instance)
get_azimuth (instance)
get_calibration (instance)
get_calibration_energy ()
get_calibration_position ()
get_calibration_wavelength ()
get_channel_count (instance)
get_collection_mode (instance)
get_crystal_2d (instance)
get_dispersion_element (instance)
get_distance (instance)
get_elevation (instance)
get_manufacturer (instance)
get_measurement_unit (instance)
get_model (instance)
get_pulse_height_analyser (instance)
get_rowland_circle_diameter (instance)
get_semi_angle (instance)
get_serial_number (instance)
get_signal_type (instance)
get_solid_angle (instance)
get_temperature (instance)
get_window (instance)
manufacturer
    manufacturer
measurement_unit
    measurement unit
model
    model
pulse_height_analyser
    pulse height analyzer
rowland_circle_diameter
    Rowland circle diameter
semi_angle
    semi-angle
serial_number
    serial number
set_area (instance, value, unit=None)
set_azimuth(instance, value, unit=None)
set_calibration(instance, value)
set_channel_count(instance, value, unit=None)
set_collection_mode(instance, value)
set_crystal_2d(instance, value, unit=None)
set_dispersion_element(instance, value)
set_distance(instance, value, unit=None)
set_elevation(instance, value, unit=None)
set_manufacturer(instance, value)
set_measurement_unit(instance, value)
set_model(instance, value)
set_pulse_height_analyser(instance, value)
set_rowland_circle_diameter(instance, value, unit=None)
set_semi_angle(instance, value, unit=None)
set_serial_number(instance, value)
set_signal_type(instance, value)
set_solid_angle(instance, value, unit=None)
set_temperature(instance, value, unit=None)
set_window(instance, value)

signal_type
type of signal

solid_angle
solid angle

temperature
temperature

window
window
class pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS(channel_count, calibration, collection_mode=None, technology=None, nominal_throughput=None, time_constant=None, strobe_rate=None, window=None, signal_type=None, manufacturer=None, model=None, serial_number=None, measurement_unit='counts', elevation=None, azimuth=None, distance=None, area=None, solid_angle=None, semi_angle=None, temperature=None)

Describes the type and configuration of an energy dispersive x-ray spectrometer.

Parameters

- **channel_count** – number of channels (required)
- **calibration** (_Calibration_) – calibration (required)
- **mode** (collection) – mode of collection, either COLLECTION_MODE_PARALLEL or COLLECTION_MODE_SERIAL (optional)
- **technology** – technology (optional)
- **nominal_throughput** – nominal throughput (optional)
- **time_constant** – time constant (optional)
- **strobe_rate** – strobe rate (optional)
- **window** (Layer) – window (optional)
- **signal_type** – type of signal (optional)
- **manufacturer** – manufacturer (optional)
- **model** – model (optional)
- **serial_number** – serial number (optional)
- **measurement_unit** – measurement unit (optional)
- **elevation** – elevation (optional)
- **azimuth** – azimuth (optional)
- **distance** – distance (optional)
- **area** – area (optional)
- **solid_angle** – solid angle (optional)
- **semi_angle** – semi-angle (optional)
- **temperature** – temperature (optional)

```python
CLASS = 'Spectrometer/XEDS'
TEMPLATE = 'Detector'

area
area

azimuth
azimuth

calibration
calibration

calibration_energy
calibration_wavelength

channel_count
number of channels

collection_mode
mode of collection

distance
distance

elevation
elevation

get_area(instance)
get_azimuth(instance)
get_calibration(instance)
get_calibration_energy()
get_calibration_wavelength()
get_channel_count(instance)
get_collection_mode(instance)
get_distance(instance)
get_elevation(instance)
get_manufacturer(instance)
get_measurement_unit(instance)
get_model(instance)
get_nominal_throughput(instance)
get_semi_angle(instance)
get_serial_number(instance)
get_signal_type(instance)
get_solid_angle(instance)
get_strobe_rate(instance)
get_technology(instance)
```
get_temperature (instance)
get_time_constant (instance)
get_window (instance)
manufacturer
  manufacturer
measurement_unit
  measurement unit
model
  model
nominal_throughput
  nominal throughput
semi_angle
  semi-angle
serial_number
  serial number
set_area (instance, value, unit=None)
set_azimuth (instance, value, unit=None)
set_calibration (instance, value)
set_channel_count (instance, value, unit=None)
set_collection_mode (instance, value)
set_distance (instance, value, unit=None)
set_elevation (instance, value, unit=None)
set_manufacturer (instance, value)
set_measurement_unit (instance, value)
set_model (instance, value)
set_nominal_throughput (instance, value, unit=None)
set_semi_angle (instance, value, unit=None)
set_serial_number (instance, value)
set_signal_type (instance, value)
set_solid_angle (instance, value, unit=None)
set_strobe_rate (instance, value, unit=None)
set_technology (instance, value)
set_temperature (instance, value, unit=None)
set_time_constant (instance, value, unit=None)
set_window (instance, value)
signal_type
  type of signal
solid_angle
  solid angle


```

class pyhmsa.spec.condition.elementalid.ElementalID:
    atomic_number=None, symbol=None

Defines and elemental identification, as may be useful for region of interest images, XAFS spectral maps, and the like.

Parameters
    atomic_number – atomic number (required)

CLASS = None

_TEMPLATE = ‘ElementalID’

atomic_number
    atomic number

get_atomic_number(self)

get_symbol()
    Returns the symbol.

set_atomic_number(self, value, unit=None)

set_symbol(symbol)

symbol
    Symbol

class pyhmsa.spec.condition.elementalid.ElementalIDXray:

Defines and elemental identification based on an x-ray peak, as may be useful for region of interest images and the like.

Parameters
    • atomic_number – atomic number (required)
    • line – x-ray line (required)
    • energy – energy of x-ray line (optional)

CLASS = ‘X-ray’
```


\[
\begin{align*}
\text{TEMPLATE} &= \text{‘ElementalID’} \\
\text{atomic_number} &\quad \text{atomic number} \\
\text{energy} &\quad \text{energy of x-ray line} \\
\text{get\_atomic\_number} (\text{instance}) \\
\text{get\_energy} (\text{instance}) \\
\text{get\_line} (\text{instance}) \\
\text{get\_symbol} () &\quad \text{Returns the symbol.} \\
\text{line} &\quad \text{x-ray line} \\
\text{set\_atomic\_number} (\text{instance}, \text{value}, \text{unit=None}) \\
\text{set\_energy} (\text{instance}, \text{value}, \text{unit=None}) \\
\text{set\_line} (\text{instance}, \text{value}, \text{notation=None}) \\
\text{set\_symbol} (\text{symbol}) \\
\text{symbol} &\quad \text{Symbol}
\end{align*}
\]

### 2.2.5 Instrument

Conditions describing the type of instrument used to collect the data.

#### Classes

```python
class pyhmsa.spec.condition.instrument.Instrument (manufacturer, model, serial_number=None)
```

Describes the type of instrument used to collect a HMSA dataset.

**Parameters**

- `manufacturer` – manufacturer (required)
- `model` – model (required)
- `serial_number` – serial number (optional)

```python
CLASS = None
TEMPLATE = ‘Instrument’
get\_manufacturer (\text{instance})
get\_model (\text{instance})
get\_serial\_number (\text{instance})
manufacturer
model
```

## 2.2. Conditions

43
serial_number
    serial number

set_manufacturer(instance, value)

set_model(instance, value)

set_serial_number(instance, value)

### 2.2.6 Probe

Conditions describing the type and conditions of the analytical probe used to collect the data.

**Constants**

**Gun types**

- `pyhmsa.spec.condition.probe.GUN_TYPE_W_FILAMENT`
- `pyhmsa.spec.condition.probe.GUN_TYPE_LAB6`
- `pyhmsa.spec.condition.probe.GUN_TYPE_COLD_FEG`
- `pyhmsa.spec.condition.probe.GUN_TYPE_SCHOTTKY_FEG`

**Lens modes**

- `pyhmsa.spec.condition.probe.LENS_MODE_IMAGE`
- `pyhmsa.spec.condition.probe.LENS_MODE_DIFFR`
- `pyhmsa.spec.condition.probe.LENS_MODE_SCIMG`
- `pyhmsa.spec.condition.probe.LENS_MODE_SCDIF`

**Classes**

```python
class pyhmsa.spec.condition.probe.ProbeEM(beam_voltage, beam_current=None, gun_type=None, emission_current=None, filament_current=None, extractor_bias=None, beam_diameter=None, chamber_pressure=None, gun_pressure=None, scan_magnification=None, working_distance=None)
```

Describes the electron column conditions of the transmission electron microscope used to collect a HMSA dataset.

**Parameters**

- **beam_voltage** – beam voltage (required)
- **beam_current** – beam current (optional)
- **gun_type** – type of gun (optional)
- **emission_current** – emission current (optional)
- **filament_current** – filament current (optional)
- **extractor_bias** – extractor bias (optional)
• **beam_diameter** – beam diameter (optional)
• **chamber_pressure** – chamber pressure (optional)
• **gun_pressure** – gun pressure (optional)
• **scan_magnification** – scan magnification (optional)
• **working_distance** – working distance (optional)

CLASS = ‘EM’

TEMPLATE = ‘Probe’

beam_current
    beam current

beam_diameter
    beam diameter

beam_voltage
    beam voltage

chamber_pressure
    chamber pressure

emission_current
    emission current

extractor_bias
    extractor bias

filament_current
    filament current

get_beam_current(*instance*)
get_beam_diameter(*instance*)
get_beam_voltage(*instance*)
get_chamber_pressure(*instance*)
get_emission_current(*instance*)
get_extractor_bias(*instance*)
get_filament_current(*instance*)
get_gun_pressure(*instance*)
get_gun_type(*instance*)
get_scan_magnification(*instance*)
get_working_distance(*instance*)

gun_pressure
    gun pressure

gun_type
    type of gun

scan_magnification
    scan magnification

set_beam_current(*instance*, *value*, *unit=None*)
```python
def set_beam_diameter(instance, value, unit=None):
    pass

def set_beam_voltage(instance, value, unit=None):
    pass

def set_chamber_pressure(instance, value, unit=None):
    pass

def set_emission_current(instance, value, unit=None):
    pass

def setExtractorBias(instance, value, unit=None):
    pass

def set_filament_current(instance, value, unit=None):
    pass

def set-gun_pressure(instance, value, unit=None):
    pass

def set-gun_type(instance, value):
    pass

def set_scan_magnification(instance, value, unit=None):
    pass

def set-working_distance(instance, value, unit=None):
    pass
```

class pyhmsa.spec.condition.probe.ProbeTEM:
    def __init__(beam_voltage, lens_mode, beam_current=None, 
                 gun_type=None, emission_current=None, filament_current=None, 
                 extractor_bias=None, beam_diameter=None, chamber_pressure=None, 
                 gun_pressure=None, scan_magnification=None, 
                 working_distance=None, camera_magnification=None, 
                 convergence_angle=None):
        pass

Describes the electron column conditions of the transmission electron microscope used to collect a HMSA dataset.

**Parameters**

- **beam_voltage** – beam voltage (required)
- **lens_mode** – lens mode (required)
- **beam_current** – beam current (optional)
- **gun_type** – gun type (optional)
- **emission_current** – emission current (optional)
- **filament_current** – filament current (optional)
- **extractor_bias** – extractor bias (optional)
- **beam_diameter** – beam diameter (optional)
- **chamber_pressure** – chamber pressure (optional)
- **gun_pressure** – gun pressure (optional)
- **scan_magnification** – scan magnification (optional)
- **working_distance** – working distance (optional)
- **camera_magnification** – camera magnification (optional)
- **convergence_angle** – semi-angle of incident beam (optional)

**CLASS** = ‘TEM’

**TEMPLATE** = ‘Probe’
beam_current
    beam current
beam_diameter
    beam diameter
beam_voltage
    beam voltage
camera_magnification
    camera magnification
chamber_pressure
    chamber pressure
convergence_angle
    semi-angle of incident beam
emission_current
    emission current
extractor_bias
    extractor bias
filament_current
    filament current
get_beam_current(instance)
get_beam_diameter(instance)
get_beam_voltage(instance)
get_camera_magnification(instance)
get_chamber_pressure(instance)
get_convergence_angle(instance)
get_emission_current(instance)
get_extractor_bias(instance)
get_filament_current(instance)
get_gun_pressure(instance)
get_gun_type(instance)
get_lens_mode(instance)
get_scan_magnification(instance)
get_working_distance(instance)
gun_pressure
    gun pressure
gun_type
    type of gun
lens_mode
    lens mode
scan_magnification
    scan magnification

2.2. Conditions
set_beam_current (instance, value, unit=None)
set_beam_diameter (instance, value, unit=None)
set_beam_voltage (instance, value, unit=None)
set_camera_magnification (instance, value, unit=None)
set_chamber_pressure (instance, value, unit=None)
set_convergence_angle (instance, value, unit=None)
set_emission_current (instance, value, unit=None)
set_extractor_bias (instance, value, unit=None)
set_filament_current (instance, value, unit=None)
set_gun_pressure (instance, value, unit=None)
set_gun_type (instance, value)
set_lens_mode (instance, value)
set_scan_magnification (instance, value, unit=None)
set_working_distance (instance, value, unit=None)

working_distance

2.2.7 Region

Conditions defining a region of spectrum.

Classes

class pyhmsa.spec.condition.region.RegionOfInterest (start_channel, end_channel)

Defines a region of a spectrum (or other one-dimensional datum), as may be useful for defining start and end channels used for a region of interest image.

Parameters

• start_channel – start channel (required)
• end_channel – end channel (required)

CLASS = None
TEMPLATE = ‘RegionOfInterest’

channels

Channel range

end_channel

End channel

get_channels (instance)

get_end_channel ()

Returns the end channel.

get_start_channel ()

Returns the start channel.
set_channels (instance, vmin, vmax, unit=None)
start_channel
    Start channel

2.2.8 Specimen

Conditions defining a physical specimen.

Helper classes

class pyhmsa.spec.condition.specimen.SpecimenLayer (name=None, thickness=None, formula=None, composition=None)

Defines a layer of a multi-layered specimen.

Parameters

• name – name (optional)
• thickness – thickness, bulk layer if None (optional)
• formula – formula
• composition – composition

composition
    composition

formula
    formula

get_composition (instance)

get_formula (instance)

get_name (instance)

get_thickness (instance)

is_bulk ()
    Returns whether this layer is a bulk layer.

name
    name

set_composition (instance, value)

set_formula (instance, value)

set_name (instance, value)

set_thickness (instance, value, unit=None)

thickness
    thickness

Classes

class pyhmsa.spec.condition.specimen.Specimen (name, description=None, origin=None, formula=None, composition=None, temperature=None)

Defines a physical specimen, including the name, origin, composition, etc.
Parameters

- **name** – name (required)
- **description** – description (optional)
- **origin** – origin (optional)
- **formula** – formula (optional)
- **composition** – composition (optional)
- **temperature** – temperature (optional)

```python
class pyhmsa.spec.condition.specimen.SpecimenMultilayer(name, description=None, origin=None, formula=None, composition=None, temperature=None, layers=None)
```

Defines a multi-layered physical specimen

Parameters

- **name** – name (required)
• **description** – description (optional)
• **origin** – origin (optional)
• **formula** – formula (optional)
• **composition** – composition (optional)
• **temperature** – temperature (optional)
• **layers** – layers (optional)

CLASS = ‘Multilayer’

TEMPLATE = ‘Specimen’

```python
append_layer(name=None, thickness=None, formula=None, composition=None)
```
Utility function to create a layer.

**Parameters**

- **name** – name (optional)
- **thickness** – thickness, bulk layer if None (optional)
- **formula** – formula
- **composition** – composition

**Returns** created layer

**Return type** SpecimenLayer

composition
composition
description
description
formula
formula

```python
get_composition(instance)
get_description(instance)
get_formula(instance)
get_layers(instance)
get_name(instance)
get_origin(instance)
get_temperature(instance)
```
layers
modifiable list of layers

```python
name
name
origin
origin
```

```python
set_composition(instance, value)
set_description(instance, value)
```
2.2.9 Specimen position

Conditions defining a physical location on (or in) the specimen.

Classes

class pyhmsa.spec.condition.specimenposition.SpecimenPosition(x=None, y=None, z=None, r=None, t=None):

Defines a physical location on (or in) the specimen. The position shall be defined in the coordinate system of the instrument. This version of the HMSA standard does not specify a template or definition of coordinate systems.

Parameters

• x – x coordinate
• y – y coordinate
• z – z coordinate
• r – rotation
• t – tilt

CLASS = None
TEMPLATE = ‘SpecimenPosition’

get_r (instance)
get_t (instance)
get_x (instance)
get_y (instance)
get_z (instance)

r
rotation

set_r (instance, value, unit=None)
set_t (instance, value, unit=None)
set_x (instance, value, unit=None)
set_y (instance, value, unit=None)
set_z (instance, value, unit=None)

t
tilt
tolist (coordinate_unit='mm', angle_unit='degrees')

\[ x \]
\[ y \]
\[ z \]

2.3 Data

List of available datasets:

2.3.1 Analysis

Dataset used to store a single measurement of a specimen at a single point in space or time.

Classes

class pyhmsa.spec.datum.analysis.Analysis0D

Data with 0 collection dimensions and 0 datum dimensions implies a dataset comprising of one single-valued measurement.

- collection_dimensions
- dimensions and order of the collections

- conditions
- Conditions associated to this dataset.

- datum_dimensions
- Dimensions and order of the data.

class pyhmsa.spec.datum.analysis.Analysis1D

Stores a measurement of a specimen at a single point in space or time with one datum dimension.

- channels

- collection_dimensions
- Dimensions and order of the collections

- conditions
- Conditions associated to this dataset.

- datum_dimensions

class pyhmsa.spec.datum.analysis.Analysis2D

Store a single measurement of the specimen at a single point in space or time with two datum dimensions, such as a diffraction pattern.

Note: This dataset type shall not be used to store 2 dimensional images rastered over the specimen, such as a conventional TEM or SEM image. Instead, such data shall be stored using the ImageRaster2D.

- collection_dimensions
- Dimensions and order of the collections
conditions
   Conditions associated to this dataset.

datum_dimensions
   u
   v

2.3.2 Analysis list

Dataset used to store a sequence of point measurements collected under the same conditions, but in an irregular pattern (line scan, time sequence, sparsely scanned image).

Classes

class pyhmsa.spec.datum.analysislist.AnalysisList0D
   Represents a sequence of point measurements with zero datum dimension, such as a line scan or time sequence of single-valued data (e.g. Ti counts, BSE yield, vacuum pressure).
   collection_dimensions
   conditions
      Conditions associated to this dataset.

datum_dimensions
   Dimensions and order of the data.

toanalysis (analysis_index)

class pyhmsa.spec.datum.analysislist.AnalysisList1D
   Represents a sequence of point measurements with one datum dimension, such as a spectrum.
   channels

   collection_dimensions
   conditions
      Conditions associated to this dataset.

datum_dimensions

toanalysis (analysis_index)

class pyhmsa.spec.datum.analysislist.AnalysisList2D
   Represents a sequence of point measurements with two datum dimensions, such as a diffraction pattern.

collection_dimensions
   conditions
      Conditions associated to this dataset.

datum_dimensions

toanalysis (analysis_index)
   u
   v
2.3.3 Image raster

Dataset used to store rastered results over regularly spaced intervals in one or more dimensions, such as a 1D linescan, a 2D image or a 3D serial section.

Classes

class pyhmsa.spec.datum.imageraster.ImageRaster2D
   Represents a dataset that has been raster mapped in 2D (x/y dimensions).
      collection_dimensions
      conditions
         Conditions associated to this dataset.
      datum_dimensions
         Dimensions and order of the data.
      toanalysis \((x, y)\)

class pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral
   Represents a dataset that has been raster mapped in 2D (x/y dimensions), where for each raster coordinate, the datum collected was a 1D array (channel dimension). An example of this type of dataset is a SEM-XEDS map.
      channels
      collection_dimensions
      conditions
         Conditions associated to this dataset.
      datum_dimensions
      toanalysis \((x, y)\)

class pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage
   Represents a dataset that has been raster mapped in 2D (x/y dimensions), where for each raster coordinate, the datum collected was a 2D image \((U/V)\) dimensions.
      collection_dimensions
      conditions
         Conditions associated to this dataset.
      datum_dimensions
      toanalysis \((x, y)\)
      \(u\)
      \(v\)

The main object of the library is the DataFile which regroups in a single object the Header, Conditions and datasets. HMSA files can be created, read and written from this object.

2.4 Data file

This is the main class to read, write and create HMSA data file.
class pyhmsa.datafile.DataFile(filepath=None, version='1.0', language='en-US')

Creates a new MSA hyper dimensional data file.

Conditions and data objects can be added using the attributes conditions and data, respectively. Note that conditions part of any datum object will also appear in the global conditions dictionary.

Parameters

- version – version of the data file (default: to most up-to-date version)
- language – language of the data file (default and recommended language is en-US)

VERSION = '1.0'

conditions

Conditions

data

Data

filepath

Path where the data file was last saved. Always .hmsa extension used.

header

Header

language

Language

merge (datafile)

orphan_conditions

Conditions that are not associated to any data sets (read-only).

classmethod read (filepath)

Reads an existing MSA hyper dimensional data file and returns an object of this class.

Parameters filepath – either the location of the XML or HMSA file. Note that both have to be present.

update (datafile)

version

Version

write (filepath=None)

Writes this data file to disk.

Parameters filepath – either the location of the XML or HMSA file

Other classes of the library used to define data types, to read and write HMSA files as well as some utilities can be found here:

2.5 Types

2.5.1 Checksum

Check sum class and method to calculate.
Constants

```python
pyhmsa.type.checksum.CHECKSUM_ALGORITHM_SHA1
pyhmsa.type.checksum.CHECKSUM_ALGORITHM_SUM32
```

Functions

```python
pyhmsa.type.checksum.calculate_checksum_sha1(buffer)
pyhmsa.type.checksum.calculate_checksum_sum32(buffer)
pyhmsa.type.checksum.calculate_checksum(algorithm, buffer)
```

Classes

```python
class pyhmsa.type.checksum.Checksum
```

2.5.2 Identifier

Base class for conditions and data dictionaries.

Functions

```python
pyhmsa.type.identifier.validate_identifier(identifier)
```

Classes

```python
class pyhmsa.type.identifier._IdentifierDict
  copy()
```

2.5.3 Language

Language type to deal with alternative spellings.

Functions

```python
pyhmsa.type.language.validate_language_tag(tag)
```

Classes

```python
class pyhmsa.type.language.langstr
  alternatives
```
2.5.4 Numerical

Special type to express magnitudes, values with a unit. It also implements the HMSA specifications restrictions on the types of data.

Functions

- `pyhmsa.type.numerical.validate_dtype(arg)`
- `pyhmsa.type.numerical.convert_value(value, unit=None)`
- `pyhmsa.type.numerical.convert_unit(newunit, value, oldunit=None)`

Classes

- `class pyhmsa.type.numerical.arrayunit`
  `unit`

2.5.5 Unique identifier id

Generate a unique identifier id.

Functions

- `pyhmsa.type.uid.generate_uid()`
  Generates a unique identifier id. The method to generate the id was taken from the C implementation of the HMSA lib.

2.5.6 Unit

Parse and validate unit.

Functions

- `pyhmsa.type.unit.parse_unit(unit)`
- `pyhmsa.type.unit.validate_unit(unit)`

2.5.7 X-ray line

X-ray line type.

Constants

- `pyhmsa.type.xrayline.NOTATION_IUPAC`
- `pyhmsa.type.xrayline.NOTATION_SIEGBAHN`
Classes

class pyhmsa.type.xrayline.xrayline

    alternative
    notation
The source code of the library can be viewed/forked/downloaded on GitHub.

### 3.1 License

Copyright (c) 2014 Philippe Pinard

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Symbols
_IdentiferDict (class in pyhmsa.type.identifier), 57

A
AcquisitionMultipoint (class in pyhmsa.spec.condition.acquisition), 15
AcquisitionPoint (class in pyhmsa.spec.condition.acquisition), 14
AcquisitionRasterLinescan (class in pyhmsa.spec.condition.acquisition), 15
AcquisitionRasterXY (class in pyhmsa.spec.condition.acquisition), 17
AcquisitionRasterXYZ (class in pyhmsa.spec.condition.acquisition), 19
alternative (pyhmsa.type.xrayline.xrayline attribute), 59
alternatives (pyhmsa.type.language.langstr attribute), 57
Analysis0D (class in pyhmsa.spec.datum.analysis), 53
Analysis1D (class in pyhmsa.spec.datum.analysis), 53
Analysis2D (class in pyhmsa.spec.datum.analysis), 53
AnalysisList0D (class in pyhmsa.spec.datum.analysislist), 54
AnalysisList1D (class in pyhmsa.spec.datum.analysislist), 54
AnalysisList2D (class in pyhmsa.spec.datum.analysislist), 54
append_layer() (pyhmsa.spec.condition.detector.Window method), 26
append_layer() (pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51
area (pyhmsa.spec.condition.detector.DetectorCamera attribute), 28
area (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 30
area (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 33
area (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 36
area (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 40
atomic_number (pyhmsa.spec.condition.elementalid.ElementalID attribute), 42
atomic_number (pyhmsa.spec.condition.elementalid.ElementalIDXray attribute), 43
author (pyhmsa.spec.header.Header attribute), 13
azimuth (pyhmsa.spec.condition.detector.DetectorCamera attribute), 28
azimuth (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 30
azimuth (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 33
azimuth (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 36
azimuth (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 40

B
base_level (pyhmsa.spec.condition.detector.PulseHeightAnalyser attribute), 27
beam_current (pyhmsa.spec.condition.probe.ProbeEM attribute), 45
beam_current (pyhmsa.spec.condition.probe.ProbeTEM attribute), 46
beam_diameter (pyhmsa.spec.condition.probe.ProbeEM attribute), 45
beam_diameter (pyhmsa.spec.condition.probe.ProbeTEM attribute), 47
beam_voltage (pyhmsa.spec.condition.probe.ProbeEM attribute), 45
beam_voltage (pyhmsa.spec.condition.probe.ProbeTEM attribute), 47
bias (pyhmsa.spec.condition.detector.PulseHeightAnalyser attribute), 27

C
calculate_checksum() (in module pyhmsa.type.checksum), 57
calculate_checksum_sha1() (in module pyhmsa.type.checksum), 57
calculate_checksum_sum32() (in module pyhmsa.type.checksum), 57
dwell_time (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

dwell_time (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

dwell_time (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

dwell_time_live (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

dwell_time_live (pyhmsa.spec.condition.acquisition.AcquisitionPoint attribute), 14

dwell_time_live (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

dwell_time_live (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

dwell_time_live (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

E

ElementalID (class in pyhmsa.spec.condition.elementalid), 42
ElementalIDXray (class in pyhmsa.spec.condition.elementalid), 42

elevation (pyhmsa.spec.condition.detector.DetectorCamera attribute), 28

elevation (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 31

elevation (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 33

elevation (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 36

elevation (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 40

emission_current (pyhmsa.spec.condition.probe.ProbeEM attribute), 45

emission_current (pyhmsa.spec.condition.probe.ProbeTEM attribute), 47

d 获取 (pyhmsa.spec.condition.composition.CompositionElemental method), 22

d_get_area() (in module pyhmsa.spec.condition.detector.DetectorCamera method), 28

d_get_area() (in module pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31

d_get_area() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 33

d_get_area() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 36

d_get_area() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40

d_get_atomic_number() (in module pyhmsa.spec.condition.elementalid.ElementalID method), 42

d_get_atomic_number() (in module pyhmsa.spec.condition.elementalid.ElementalIDXray method), 43

d_get_azimuth() (in module pyhmsa.spec.condition.detector.DetectorCamera method), 28

d_get_azimuth() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 31

d_get_azimuth() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 36

d_get_azimuth() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40

d_get_base_level() (in module pyhmsa.spec.condition.detector.PulseHeightAnalyser method), 27

F

filament_current (pyhmsa.spec.condition.probe.ProbeEM attribute), 45

d_get_base_level() (in module pyhmsa.spec.condition.detector.PulseHeightAnalyser method), 27

d_get_beam_current() (in module pyhmsa.spec.condition.probe.ProbeEM method), 45

G

gain (pyhmsa.spec.condition.calibration.CalibrationLinear attribute), 24

gain (pyhmsa.spec.condition.detector.PulseHeightAnalyser attribute), 27

generate_uid() (in module pyhmsa.type.uid), 58

get() (in module pyhmsa.spec.condition.composition.CompositionElemental method), 22

get_area() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 36

get_area() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40

g_get_atomic_number() (in module pyhmsa.spec.condition.elementalid.ElementalID method), 42

g_get_atomic_number() (in module pyhmsa.spec.condition.elementalid.ElementalIDXray method), 43

g_get_azimuth() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 36

g_get_azimuth() (in module pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40

H

h (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

I

image (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

J

j (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

K

k (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

L

length (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

M

mass (pyhmsa.spec.condition.calibration.CalibrationPolynomial attribute), 24

N

c (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

c (pyhmsa.spec.condition.acquisition.AcquisitionPoint attribute), 14

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

c (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

c (pyhmsa.spec.condition.acquisition.AcquisitionPoint attribute), 14

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

c (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

c (pyhmsa.spec.condition.acquisition.AcquisitionPoint attribute), 14

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

c (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

c (pyhmsa.spec.condition.acquisition.AcquisitionPoint attribute), 14

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

c (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

O

offset (pyhmsa.spec.condition.detector.DetectorCamera attribute), 28

P

pose (pyhmsa.spec.condition.detector.DetectorCamera attribute), 28

Q

q (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

R

r (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

S

sensor (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 31

T

t (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

U

u (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

V

v (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

W

w (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

X

x (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

Y

y (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

Z

z (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16
get_dwell_time_live() (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint method), 15
get_dwell_time_live() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 14
get_dwell_time_live() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan method), 16
get_dwell_time_live() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18
get_dwell_time_live() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20
get_elevation() (pyhmsa.spec.condition.detector.DetectorCamera method), 28
get_elevation() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31
get_elevation() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34
get_elevation() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37
get_elevation() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40
get_emission_current() (pyhmsa.spec.condition.probe.ProbeEM method), 45
get_emission_current() (pyhmsa.spec.condition.probe.ProbeTEM method), 47
get_end_channel() (pyhmsa.spec.condition.region.RegionOfInterest method), 48
get_energy() (pyhmsa.spec.condition.elementalid.ElementalIDXray method), 43
get_exposure_time() (pyhmsa.spec.condition.detector.DetectorCamera method), 28
get_extractor_bias() (pyhmsa.spec.condition.probe.ProbeEM method), 45
get_extractor_bias() (pyhmsa.spec.condition.probe.ProbeTEM method), 47
get_filament_current() (pyhmsa.spec.condition.probe.ProbeEM method), 45
get_filament_current() (pyhmsa.spec.condition.probe.ProbeTEM method), 47
get_focal_length() (pyhmsa.spec.condition.detector.DetectorCamera method), 28
get_formula() (pyhmsa.spec.condition.specimen.Specimen method), 50
get_formula() (pyhmsa.spec.condition.specimen.SpecimenLayer method), 49
get_formula() (pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51
get_frame_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan method), 16
get_frame_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18
get_frame_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20
get_gain() (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24
get_gain() (pyhmsa.spec.condition.detector.PulseHeightAnalyser method), 27
get_grating_d() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34
get_gun_pressure() (pyhmsa.spec.condition.probe.ProbeEM method), 45
get_gun_pressure() (pyhmsa.spec.condition.probe.ProbeTEM method), 47
get_gun_type() (pyhmsa.spec.condition.probe.ProbeEM method), 45
get_gun_type() (pyhmsa.spec.condition.probe.ProbeTEM method), 47
get_index() (pyhmsa.spec.condition.calibration.CalibrationConstant method), 23
get_index() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25
get_index() (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24
get_index() (pyhmsa.spec.condition.calibration.CalibrationPolynomial method), 25
get_label() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25
get_labels() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25
get_layers() (pyhmsa.spec.condition.detector.Window method), 26
get_layers() (pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51
get_measurement_unit() (pyhmsa.spec.condition.detector.DetectorCamera method), 28
get_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31
get_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34
get_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37
<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyhmsa.spec.condition.probe.GUN_TYPE_W_FILAMENT</td>
<td>44</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.LENS_MODE_DIFFR</td>
<td>44</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.LENS_MODE_IMAGE</td>
<td>44</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.LENS_MODE_SCDIF</td>
<td>44</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.LENS_MODE_SCIIMG</td>
<td>44</td>
</tr>
<tr>
<td>pyhmsa.type.checksum.CHECKSUM_ALGORITHM_SHA1</td>
<td>57</td>
</tr>
<tr>
<td>pyhmsa.type.checksum.CHECKSUM_ALGORITHM_SUM32</td>
<td>57</td>
</tr>
<tr>
<td>pyhmsa.type.xrayline.NOTATION_IUPAC</td>
<td>58</td>
</tr>
<tr>
<td>pyhmsa.type.xrayline.NOTATION_SIEGBAHN</td>
<td>58</td>
</tr>
<tr>
<td>Quantity:</td>
<td>23, 25, 24</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.calibration.CalibrationConstant</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.calibration.CalibrationExplicit</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.calibration.CalibrationLinear</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.calibration.CalibrationPolynomial</td>
<td></td>
</tr>
<tr>
<td>Rowland circle diameter:</td>
<td>37</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometer</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerCL</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerWDS</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS</td>
<td></td>
</tr>
<tr>
<td>Serial number:</td>
<td>29, 31, 34, 37, 41</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorCamera</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerWDS</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS</td>
<td></td>
</tr>
<tr>
<td>Semi angle:</td>
<td>29, 31, 34, 37, 41</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerCL</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerWDS</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS</td>
<td></td>
</tr>
<tr>
<td>Scan magnification:</td>
<td>45, 47</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.ProbeEM</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.ProbeTEM</td>
<td></td>
</tr>
<tr>
<td>Region of Interest:</td>
<td>56</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.region.RegionOfInterest</td>
<td></td>
</tr>
<tr>
<td>Rowland circle diameter:</td>
<td>37</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometer</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerWDS</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS</td>
<td></td>
</tr>
<tr>
<td>Set area:</td>
<td>29, 32, 34, 37, 41</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorCamera</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerWDS</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS</td>
<td></td>
</tr>
<tr>
<td>Set atomic number:</td>
<td>42, 43</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.elementalid.ElementalID</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.elementalid.ElementalIDXray</td>
<td></td>
</tr>
<tr>
<td>Set azimuth:</td>
<td>29, 32, 34, 37, 41</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorCamera</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerWDS</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS</td>
<td></td>
</tr>
<tr>
<td>Set base level:</td>
<td>27</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.detector.PulseHeightAnalyser</td>
<td></td>
</tr>
<tr>
<td>Set beam current:</td>
<td>45, 47</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.ProbeEM</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.ProbeTEM</td>
<td></td>
</tr>
<tr>
<td>Set beam diameter:</td>
<td>45, 48</td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.ProbeEM</td>
<td></td>
</tr>
<tr>
<td>pyhmsa.spec.condition.probe.ProbeTEM</td>
<td></td>
</tr>
</tbody>
</table>

Index 73
Index
set_extractor_bias() (pyhmsa.spec.condition.probe.ProbeTEM
method), 48
set_filament_current() (pyhmsa.spec.condition.probe.ProbeEM
method), 46
set_filament_current() (pyhmsa.spec.condition.probe.ProbeTE
method), 48
set_focal_length() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
set_formula() (pyhmsa.spec.condition.specimen.Specimen
method), 50
set_formula() (pyhmsa.spec.condition.specimen.SpecimenLayer
method), 49
set_formula() (pyhmsa.spec.condition.specimen.SpecimenMultilayer
method), 51
set_frame_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinesca
method), 17
set_frame_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY
method), 18
set_gain() (pyhmsa.spec.condition.calibration.CalibrationLinear
method), 24
set_gain() (pyhmsa.spec.condition.detector.PulseHeightAnalyser
method), 27
set_grating_d() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
method), 34
set_gun_pressure() (pyhmsa.spec.condition.probe.ProbeEM
method), 46
set_gun_pressure() (pyhmsa.spec.condition.probe.ProbeTEM
method), 48
set_gun_type() (pyhmsa.spec.condition.probe.ProbeEM
method), 46
set_gun_type() (pyhmsa.spec.condition.probe.ProbeTEM
method), 48
set_labels() (pyhmsa.spec.condition.calibration.CalibrationExplicit
method), 25
set_lens_mode() (pyhmsa.spec.condition.probe.ProbeTEM
method), 48
set_line() (pyhmsa.spec.condition.elementalid.ElementalIDXray
method), 43
set_magnification() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
set_manufacturer() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
set_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometer
method), 32
set_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
method), 34
set_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
method), 38
set_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
method), 41
set_mode() (pyhmsa.spec.condition.detector.PulseHeightAnalyser
method), 27
set_model() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
set_model() (pyhmsa.spec.condition.detector.DetectorSpectrometer
method), 32
set_model() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
method), 34
set_model() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
method), 38
set_model() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
method), 41
set_material() (pyhmsa.spec.condition.detector.WindowLayer
method), 26
set_measurement_unit() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
set_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometer
method), 32
set_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
method), 34
set_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
method), 38
set_measurement_unit() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
method), 41
set_offset() (pyhmsa.spec.condition.calibration.CalibrationLinear
method), 24
set_origin() (pyhmsa.spec.condition.specimen.Specimen
method), 50
set_origin() (pyhmsa.spec.condition.specimen.SpecimenLayer
method), 49
set_wrong() (pyhmsa.spec.condition.specimen.SpecimenMultilayer
method), 52
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionPoint
method), 14
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY
method), 18
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
method), 20
set_position_end() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinesca
method), 17
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinesca
method), 17
set_pixel_count_u() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
set_pixel_count_v() (pyhmsa.spec.condition.detector.DetectorCamera
method), 29
setxEC() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
method), 38
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
method), 18
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
method), 18
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
method), 18
set_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
method), 18
set_pulse_height_analyser()
set_quantity() (pyhmsa.spec.condition.calibration.CalibrationConstant method), 35
set_quantity() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25
set_quantity() (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24
set_quantity() (pyhmsa.spec.condition.calibration.CalibrationPolynomial method), 25
set_r() (pyhmsa.spec.condition.specimenposition.SpecimenPosition method), 52
set_raster_mode() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan method), 17
set_raster_mode() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_raster_mode() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 21
set_raster_mode_z() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 21
set_rowland_circle_diameter() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 38
set_scan_magnification() (pyhmsa.spec.condition.probe.ProbeEM method), 46
set_scan_magnification() (pyhmsa.spec.condition.probe.ProbeTEM method), 48
set_semi_angle() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32
set_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 35
set_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 38
set_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_serial_number() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_serial_number() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32
set_serial_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 35
set_serial_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 38
set_serial_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_serial_number() (pyhmsa.spec.condition.instrument.Instrument method), 44
set_solid_angle() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_solid_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32
set_solid_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 35
set_solid_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 38
set_solid_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_step_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan method), 17
set_step_count_x() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_step_count_y() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_step_count_z() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 21
set_step_size() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan method), 17
set_step_size_x() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_step_size_y() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_step_size_z() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 21
set_strobe_rate() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_symbol() (pyhmsa.spec.condition.elementalid.ElementalID method), 42
set_symbol() (pyhmsa.spec.condition.elementalid.ElementalIDXray method), 43
set_t() (pyhmsa.spec.condition.specimenposition.SpecimenPosition method), 52
set_technology() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_temperature() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_temperature() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32
set_temperature() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 35
set_temperature() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 38
set_temperature() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_step_size() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan method), 17
set_step_size_x() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_step_size_y() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 19
set_step_size_z() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 21
set_strobe_rate() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_symbol() (pyhmsa.spec.condition.elementalid.ElementalID method), 42
set_symbol() (pyhmsa.spec.condition.elementalid.ElementalIDXray method), 43
set_t() (pyhmsa.spec.condition.specimenposition.SpecimenPosition method), 52
set_technology() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 41
set_temperature() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
attribute), 19
step_size_y (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY
attribute), 21
step_size_z (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
attribute), 21
strobe_rate (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
attribute), 41
symbol (pyhmsa.spec.condition.elementalid.ElementalID
attribute), 42
symbol (pyhmsa.spec.condition.elementalid.ElementalIDXray
attribute), 43
T
T (pyhmsa.spec.condition.specimenposition.SpecimenPosition
attribute), 52
technology (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
attribute), 42
temperature (pyhmsa.spec.condition.detector.DetectorCamera
attribute), 30
temperature (pyhmsa.spec.condition.detector.DetectorSpectrometer
attribute), 32
temperature (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
attribute), 35
temperature (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
attribute), 38
temperature (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
attribute), 42
temperature (pyhmsa.spec.condition.specimen.Specimen
attribute), 50
temperature (pyhmsa.spec.condition.specimen.SpecimenMultilayer
attribute), 51
time (pyhmsa.spec.header.Header attribute), 13
time_constant (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
attribute), 42
time (pyhmsa.spec.header.Header attribute), 13
to_wt() (pyhmsa.spec.condition.composition.CompositionElemental
method), 22
toanalysis() (pyhmsa.spec.datum.analysislist.AnalysisList0D
method), 54
toanalysis() (pyhmsa.spec.datum.analysislist.AnalysisList1D
method), 54
toanalysis() (pyhmsa.spec.datum.analysislist.AnalysisList2D
method), 54
toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2D
method), 55
toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage
method), 55
toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral
method), 55
toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral
method), 55
toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral
method), 55
toanalysis() (pyhmsa.spec.datum.specimenposition.SpecimenPosition
method), 52
total_time (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint
attribute), 15
total_time (pyhmsa.spec.condition.acquisition.AcquisitionPoint
attribute), 15
total_time (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan
attribute), 17
total_time (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
attribute), 19
total_time (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ
attribute), 21