

---

# **pyHMSA Documentation**

***Release 0.1.6***

**Philippe Pinard**

January 17, 2015



<b>1</b>	<b>Examples</b>	<b>3</b>
1.1	Create a new data file . . . . .	3
1.2	Read a data file . . . . .	6
1.3	Assign a condition to a data set . . . . .	7
1.4	Extract a spectrum to a CSV file . . . . .	9
1.5	Use alternative language . . . . .	10
1.6	Create an Elemental ID x-ray condition . . . . .	11
<b>2</b>	<b>API</b>	<b>13</b>
2.1	Header . . . . .	13
2.2	Conditions . . . . .	13
2.3	Data . . . . .	52
2.4	Data file . . . . .	55
2.5	Types . . . . .	56
<b>3</b>	<b>Download</b>	<b>59</b>
3.1	License . . . . .	59



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](#).



---

## Examples

---

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:

```
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:

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

is equivalent to this:

```
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.

```
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`.

```
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 (`dwelt_time`, `total_time`, `dwelt_time_live`) can be defined later.

```
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.

```
acq.set_dwelt_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.

```
acq.dwelt_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.

```
acq.dwelt_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,

```
print(acq.dwelt_time) # Returns: 5.0 s
print(acq.get_dwelt_time()) # Returns: 5.0 s
print(acq.dwelt_time.unit) # Returns: s
```

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

```
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.

```
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.

```
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 has 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

```
#!/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')
acq.dwell_time = (5.0, 's')
acq.dwell_time = 5.0

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

```
## 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.

```
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`.

```
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:

```
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

```
#!/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*):

```
# 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.

```
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.

```
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 modified the same condition from two locations: the data file or the data set.

```
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.

```
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.

```
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

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

from 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.

```

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.

```
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:

```

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.

```

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.

```
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

```
#!/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.

```
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.

```
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.

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

### 1.5.1 Full source code

```
#!/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-L3*) and the traditional Siegbahn notation (*Kα1*). 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 *Kα1* line as follows:

```
from pyhmsa.type.xrayline import xrayline, NOTATION_SIEGBAHN
line = xrayline('Kα1', 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.

```
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=Kα1)>
```

### 1.6.1 Full source code

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

from pyhmsa.type.xrayline import xrayline, NOTATION_SIEGBAHN
line = xrayline('K $\alpha$ 1', 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=K $\alpha$ 1)>
```



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

```
pyhmsa.spec.condition.acquisition.RASTER_MODE_STAGE
```

```
pyhmsa.spec.condition.acquisition.RASTER_MODE_BEAM
```

### Z raster modes

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

### Position locations

```
pyhmsa.spec.condition.acquisition.POSITION_LOCATION_START
```

```
pyhmsa.spec.condition.acquisition.POSITION_LOCATION_CENTER
```

```
pyhmsa.spec.condition.acquisition.POSITION_LOCATION_END
```

## Classes

```
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** (*instance*)

**get\_dwell\_time\_live** (*instance*)

**get\_position** (*instance*)

**get\_total\_time** (*instance*)

**position**

physical location on (or in) the specimen

**set\_dwell\_time** (*instance*, *value*, *unit=None*)

**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

```
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

**get\_dwell\_time** (*instance*)

**get\_dwell\_time\_live** (*instance*)

**get\_positions** (*instance*)

**get\_total\_time** (*instance*)

**positions**

specimen positions

**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

```
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)
- **dwelt\_time** – uniform real time taken for each individual measurement (optional)
- **total\_time** – total real time taken to collect all measurements (optional)
- **dwelt\_time\_live** – analogous detector live time for each individual measurement (optional)

**CLASS** = 'Raster/XY'

**TEMPLATE** = 'Acquisition'

**dwelt\_time**

uniform real time taken for each individual measurement

**dwelt\_time\_live**

analogous detector live time for each individual measurement

**frame\_count**

number of accumulated frames

**get\_dwelt\_time** (*instance*)

**get\_dwelt\_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\_dwelt\_time** (*instance, value, unit=None*)

**set\_dwelt\_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`

**set\_raster\_mode** (*instance, value*)

**set\_step\_count\_x** (*instance, value, unit=None*)

**set\_step\_count\_y** (*instance, value, unit=None*)

**set\_step\_size\_x** (*instance, value, unit=None*)

**set\_step\_size\_y** (*instance, value, unit=None*)

**set\_total\_time** (*instance, value, unit=None*)

**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

```
class pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ (step_count_x,
                                                                step_count_y,
                                                                step_count_z,
                                                                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)
```

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)
- **dwelt\_time** – uniform real time taken for each individual measurement (optional)
- **total\_time** – total real time taken to collect all measurements (optional)
- **dwelt\_time\_live** – analogous detector live time for each individual measurement (optional)

**CLASS** = 'Raster/XYZ'

**TEMPLATE** = 'Acquisition'

**dwelt\_time**

uniform real time taken for each individual measurement

**dwelt\_time\_live**

analogous detector live time for each individual measurement

**get\_dwelt\_time** (*instance*)

**get\_dwelt\_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

**set\_dwelt\_time** (*instance, value, unit=None*)

**set\_dwelt\_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`

**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*)

**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

**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)

**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** (*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 `KeyError` is raised.

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

**set\_unit** (*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 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)

**func**

**gain**  
gain

**get\_gain** (*instance*)

**get\_index** (*value*)

**get\_offset** (*instance*)

**get\_quantity** (*index*)

**get\_unit** (*instance*)

**offset**  
offset

**quantity**  
physical quantity

**set\_gain** (*instance, value, unit=None*)

**set\_offset** (*instance, value, unit=None*)

**set\_quantity** (*instance, value*)

**set\_unit** (*instance, value*)

**unit**  
unit

**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

```

func
get_coefficients (instance)
get_index (value)
get_quantity (index)
get_unit (instance)
quantity
    physical quantity
set_coefficients (instance, value, unit=None)
set_quantity (instance, value)
set_unit (instance, value)
unit
    unit

```

**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
    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, value)

set_window (instance, value, unit=None)

window
    window

```

## Classes

```

class pyhmsa.spec.condition.detector.DetectorCamera (pixel_count_u,      pixel_count_v,
                                                    exposure_time=None,  magnifica-
                                                    tion=None,      focal_length=None,
                                                    signal_type=None,    manufac-
                                                    turer=None,      model=None,
                                                    serial_number=None,  mea-
                                                    surement_unit='counts',
                                                    elevation=None,      az-
                                                    imuth=None,      distance=None,
                                                    area=None,      solid_angle=None,
                                                    semi_angle=None,    tempera-
                                                    ture=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**  
    magnification  
**manufacturer**  
    manufacturer  
**measurement\_unit**  
    measurement unit  
**model**  
    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

**channel\_count**  
number of channels

**collection\_mode**  
mode of collection

**distance**  
distance

**elevation**  
elevation

**get\_area** (*instance*)

**get\_azimuth** (*instance*)

**get\_calibration** (*instance*)

**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

**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**

area

**azimuth**

azimuth

**calibration**

calibration

**channel\_count**

number of channels

**collection\_mode**

mode of collection

**distance**

distance

**elevation**

elevation

**get\_area** (*instance*)

**get\_azimuth** (*instance*)

**get\_calibration** (*instance*)

**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

**channel\_count**

number of channels

**collection\_mode**

mode of collection

**crystal\_2d**

crystal 2d-spacing

**dispersion\_element**

dispersion element

**distance**

distance

**elevation**

elevation

**get\_area** (*instance*)

**get\_azimuth** (*instance*)

**get\_calibration** (*instance*)

**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, collec-  
                                                                tion_mode=None,  
                                                                technol-  
                                                                ogy=None, nomi-  
                                                                nal_throughput=None,  
                                                                time_constant=None,  
                                                                strobe_rate=None,  
                                                                window=None,  
                                                                signal_type=None,  
                                                                manufacturer=None,  
                                                                model=None, se-  
                                                                rial_number=None,  
                                                                measure-  
                                                                ment_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)

**CLASS** = 'Spectrometer/XEDS'

**TEMPLATE** = 'Detector'

**area**  
area

**azimuth**  
azimuth

**calibration**  
calibration

**channel\_count**  
number of channels

**collection\_mode**  
mode of collection

**distance**  
distance

**elevation**  
elevation

**get\_area** (*instance*)

**get\_azimuth** (*instance*)

**get\_calibration** (*instance*)

**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

strobe_rate
    strobe rate

technology
    technology

temperature
    temperature

time_constant
    time constant

window
    window

```

## 2.2.4 Elemental ID

Conditions defining elemental identification.

### Classes

```

class pyhmsa.spec.condition.elementalid.ElementalID (atomic_number=None, sym-
                                                    bol=None)
    Defines and elemental identification, as may be useful for region of interest images, XAFS spectral maps, and
    the like.

    Parameters z – atomic number (required)

    CLASS = None

    TEMPLATE = 'ElementalID'

    atomic_number
        atomic number

    get_atomic_number (instance)

    get_symbol ()
        Returns the symbol.

    set_atomic_number (instance, value, unit=None)

    set_symbol (symbol)

    symbol
        Symbol

```

```
class pyhmsa.spec.condition.elementalid.ElementalIDXray (atomic_number=None,  
                                                         line=None,    energy=None,  
                                                         symbol=None)
```

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'

**TEMPLATE** = 'ElementalID'

**atomic\_number**  
atomic number

**energy**  
energy of x-ray line

**get\_atomic\_number** (*instance*)

**get\_energy** (*instance*)

**get\_line** (*instance*)

**get\_symbol** ()  
Returns the symbol.

**line**  
x-ray line

**set\_atomic\_number** (*instance*, *value*, *unit=None*)

**set\_energy** (*instance*, *value*, *unit=None*)

**set\_line** (*instance*, *value*, *notation=None*)

**set\_symbol** (*symbol*)

**symbol**  
Symbol

## 2.2.5 Instrument

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

### Classes

```
class pyhmsa.spec.condition.instrument.Instrument (manufacturer,    model,    se-  
                                                    rial_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)

```
CLASS = None
TEMPLATE = 'Instrument'
get_manufacturer (instance)
get_model (instance)
get_serial_number (instance)
manufacturer
    manufacturer
model
    model
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

```
class pyhmsa.spec.condition.probe.ProbeEM(beam_voltage,          beam_current=None,
                                           gun_type=None,   emission_current=None, fil-
                                           ament_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)
set_beam_diameter (instance, value, unit=None)
set_beam_voltage (instance, value, unit=None)
set_chamber_pressure (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_scan_magnification (instance, value, unit=None)
set_working_distance (instance, value, unit=None)

working_distance
    working_distance

```

```

class pyhmsa.spec.condition.probe.ProbeTEM(beam_voltage, lens_mode, beam_current=None,
                                           gun_type=None, emission_current=None, fil-
                                           ament_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)

```

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

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
    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)

**CLASS** = None

**TEMPLATE** = 'Specimen'

```

composition
    composition

```

```

description
    description

```

```

formula
    formula

```

```

get_composition (instance)

```

```

get_description (instance)

```

```

get_formula (instance)

```

```

get_name (instance)

```

```

get_origin (instance)

```

```

get_temperature (instance)

```

```

name
    name

```

```

origin
    origin

```

```
set_composition(instance, value)
set_description(instance, value)
set_formula(instance, value)
set_name(instance, value)
set_origin(instance, value)
set_temperature(instance, value, unit=None)
temperature
    temperature
```

```
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'

**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

**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
name
    name
origin
    origin
set_composition (instance, value)
set_description (instance, value)
set_formula (instance, value)
set_name (instance, value)
set_origin (instance, value)
set_temperature (instance, value, unit=None)
temperature
    temperature

```

## 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  
**x**  
    x coordinate  
**y**  
    y coordinate  
**z**  
    z coordinate

## 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

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

#### Functions

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

#### Classes

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

### 2.5.2 Identifier

Base class for conditions and data dictionaries.

#### Functions

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

#### Classes

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

### 2.5.3 Language

Language type to deal with alternative spellings.

#### Functions

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

## Classes

`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`

---

**Download**

---

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

`_IdentifierDict` (class in `pyhmsa.type.identifier`), 56

## A

`AcquisitionMultipoint` (class  
    `pyhmsa.spec.condition.acquisition`), 15

`AcquisitionPoint` (class  
    `pyhmsa.spec.condition.acquisition`), 14

`AcquisitionRasterLinescan` (class  
    `pyhmsa.spec.condition.acquisition`), 15

`AcquisitionRasterXY` (class  
    `pyhmsa.spec.condition.acquisition`), 17

`AcquisitionRasterXYZ` (class  
    `pyhmsa.spec.condition.acquisition`), 19

`alternative` (`pyhmsa.type.xrayline.xrayline` attribute), 58

`alternatives` (`pyhmsa.type.language.langstr` attribute), 57

`Analysis0D` (class in `pyhmsa.spec.datum.analysis`), 52

`Analysis1D` (class in `pyhmsa.spec.datum.analysis`), 52

`Analysis2D` (class in `pyhmsa.spec.datum.analysis`), 53

`AnalysisList0D` (class  
    `pyhmsa.spec.datum.analysislist`), 53

`AnalysisList1D` (class  
    `pyhmsa.spec.datum.analysislist`), 53

`AnalysisList2D` (class  
    `pyhmsa.spec.datum.analysislist`), 53

`append_layer()` (`pyhmsa.spec.condition.detector.Window`  
    method), 26

`append_layer()` (`pyhmsa.spec.condition.specimen.SpecimenMultilayer`  
    method), 50

`area` (`pyhmsa.spec.condition.detector.DetectorCamera` at-  
    tribute), 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), 39

`arrayunit` (class in `pyhmsa.type.numerical`), 57

`atomic_number` (`pyhmsa.spec.condition.elementalid.ElementalID`  
    attribute), 41

`atomic_number` (`pyhmsa.spec.condition.elementalid.ElementalIDXray`  
    attribute), 42

in `author` (`pyhmsa.spec.header.Header` attribute), 13

in `azimuth` (`pyhmsa.spec.condition.detector.DetectorCamera`  
    attribute), 28

in `azimuth` (`pyhmsa.spec.condition.detector.DetectorSpectrometer`  
    attribute), 30

in `azimuth` (`pyhmsa.spec.condition.detector.DetectorSpectrometerCL`  
    attribute), 33

in `azimuth` (`pyhmsa.spec.condition.detector.DetectorSpectrometerWDS`  
    attribute), 36

in `azimuth` (`pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS`  
    attribute), 39

## B

`base_level` (`pyhmsa.spec.condition.detector.PulseHeightAnalyser`  
    attribute), 27

in `beam_current` (`pyhmsa.spec.condition.probe.ProbeEM`  
    attribute), 44

in `beam_current` (`pyhmsa.spec.condition.probe.ProbeTEM`  
    attribute), 46

in `beam_diameter` (`pyhmsa.spec.condition.probe.ProbeEM`  
    attribute), 44

`beam_diameter` (`pyhmsa.spec.condition.probe.ProbeTEM`  
    attribute), 46

`beam_voltage` (`pyhmsa.spec.condition.probe.ProbeEM`  
    attribute), 44

`beam_voltage` (`pyhmsa.spec.condition.probe.ProbeTEM`  
    attribute), 46

`bias` (`pyhmsa.spec.condition.detector.PulseHeightAnalyser`  
    attribute), 27

## C

`calculate_checksum()` (in  
    `pyhmsa.type.checksum`), 56 module

`calculate_checksum_shal()` (in  
    `pyhmsa.type.checksum`), 56 module

`calculate_checksum_sum32()` (in  
    `pyhmsa.type.checksum`), 56 module

calibration (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASS (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 31 attribute), 30

calibration (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASS (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 33 attribute), 33

calibration (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASSWDS (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 36 attribute), 36

calibration (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASSXEDS (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 39 attribute), 39

CalibrationConstant (class in CLASS (pyhmsa.spec.condition.elementalid.ElementalID attribute), 23 attribute), 41

CalibrationExplicit (class in CLASS (pyhmsa.spec.condition.elementalid.ElementalIDXray attribute), 25 attribute), 42

CalibrationLinear (class in CLASS (pyhmsa.spec.condition.instrument.Instrument attribute), 24 attribute), 42

CalibrationPolynomial (class in CLASS (pyhmsa.spec.condition.probe.ProbeEM attribute), 24 attribute), 44

camera\_magnification (pyhmsa.spec.condition.probe.ProbeTEMGLASS (pyhmsa.spec.condition.probe.ProbeTEM attribute), 46 attribute), 46

chamber\_pressure (pyhmsa.spec.condition.probe.ProbeEM CLASS (pyhmsa.spec.condition.region.RegionOfInterest attribute), 44 attribute), 48

chamber\_pressure (pyhmsa.spec.condition.probe.ProbeTEMCLASS (pyhmsa.spec.condition.specimen.Specimen attribute), 46 attribute), 49

channel\_count (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASS (pyhmsa.spec.condition.specimen.SpecimenMultilayer attribute), 31 attribute), 50

channel\_count (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASS (pyhmsa.spec.condition.specimenposition.SpecimenPosition attribute), 33 attribute), 51

channel\_count (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASSWDS (pyhmsa.spec.condition.composition.CompositionElemental attribute), 36 method), 21

channel\_count (pyhmsa.spec.condition.detector.DetectorSpectrometerGLASSXEDS (pyhmsa.spec.condition.calibration.CalibrationPolynomial attribute), 39 attribute), 24

channels (pyhmsa.spec.condition.region.RegionOfInterest collection\_dimensions (pyhmsa.spec.datum.analysis.Analysis0D attribute), 48 attribute), 52

channels (pyhmsa.spec.datum.analysis.Analysis1D attribute\_dimensions (pyhmsa.spec.datum.analysis.Analysis1D attribute), 52 attribute), 52

channels (pyhmsa.spec.datum.analysislist.AnalysisList1D collection\_dimensions (pyhmsa.spec.datum.analysis.Analysis2D attribute), 53 attribute), 53

channels (pyhmsa.spec.datum.imageraster.ImageRaster2DSpecimen collection\_dimensions (pyhmsa.spec.datum.analysislist.AnalysisList0D attribute), 54 attribute), 53

Checksum (class in pyhmsa.type.checksum), 56 collection\_dimensions (pyhmsa.spec.datum.analysislist.AnalysisList1D attribute), 53

checksum (pyhmsa.spec.header.Header attribute), 13 attribute), 53

CLASS (pyhmsa.spec.condition.acquisition.AcquisitionMultilayer collection\_dimensions (pyhmsa.spec.datum.analysislist.AnalysisList2D attribute), 15 attribute), 54

CLASS (pyhmsa.spec.condition.acquisition.AcquisitionPoint collection\_dimensions (pyhmsa.spec.datum.imageraster.ImageRaster2D attribute), 14 attribute), 54

CLASS (pyhmsa.spec.condition.acquisition.AcquisitionRaster collection\_dimensions (pyhmsa.spec.datum.imageraster.ImageRaster2DHY attribute), 16 attribute), 54

CLASS (pyhmsa.spec.condition.acquisition.AcquisitionRaster collection\_dimensions (pyhmsa.spec.datum.imageraster.ImageRaster2DSpec attribute), 18 attribute), 54

CLASS (pyhmsa.spec.condition.acquisition.AcquisitionRaster collection\_dimensions (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 20 attribute), 31

CLASS (pyhmsa.spec.condition.composition.CompositionElemental collection\_dimensions (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 21 attribute), 33

CLASS (pyhmsa.spec.condition.detector.DetectorCamera collection\_mode (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 28 attribute), 36

collection\_mode (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 39

composition (pyhmsa.spec.condition.specimen.Specimen attribute), 49

composition (pyhmsa.spec.condition.specimen.SpecimenLayer attribute), 48

composition (pyhmsa.spec.condition.specimen.SpecimenMultilayer attribute), 50

CompositionElemental (class in pyhmsa.spec.condition.composition), 21

conditions (pyhmsa.datafile.DataFile attribute), 55

conditions (pyhmsa.spec.datum.analysis.Analysis0D attribute), 52

conditions (pyhmsa.spec.datum.analysis.Analysis1D attribute), 53

conditions (pyhmsa.spec.datum.analysis.Analysis2D attribute), 53

conditions (pyhmsa.spec.datum.analysislist.AnalysisList0D attribute), 53

conditions (pyhmsa.spec.datum.analysislist.AnalysisList1D attribute), 53

conditions (pyhmsa.spec.datum.analysislist.AnalysisList2D attribute), 54

conditions (pyhmsa.spec.datum.imageraster.ImageRaster2D attribute), 54

conditions (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage attribute), 54

conditions (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral attribute), 54

convergence\_angle (pyhmsa.spec.condition.probe.ProbeTEM attribute), 46

convert\_unit() (in module pyhmsa.type.numerical), 57

convert\_value() (in module pyhmsa.type.numerical), 57

copy() (pyhmsa.type.identifier.IdentifierDict method), 56

crystal\_2d (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 36

datum\_dimensions (pyhmsa.spec.datum.imageraster.ImageRaster2D attribute), 54

datum\_dimensions (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage attribute), 54

datum\_dimensions (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral attribute), 54

description (pyhmsa.spec.condition.specimen.SpecimenMultilayer attribute), 50

DetectorCamera (class in pyhmsa.spec.condition.detector), 27

DetectorSpectrometer (class in pyhmsa.spec.condition.detector), 30

DetectorSpectrometerCL (class in pyhmsa.spec.condition.detector), 32

DetectorSpectrometerWDS (class in pyhmsa.spec.condition.detector), 35

DetectorSpectrometerXEDS (class in pyhmsa.spec.condition.detector), 38

dispersion\_element (pyhmsa.spec.condition.detector.DetectorSpectrometerV attribute), 36

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

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

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

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

distance (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 39

dwell\_time (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint attribute), 15

dwell\_time (pyhmsa.spec.condition.acquisition.AcquisitionPoint attribute), 14

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

data (pyhmsa.datafile.DataFile attribute), 55

DataFile (class in pyhmsa.datafile), 55

date (pyhmsa.spec.header.Header attribute), 13

datum\_dimensions (pyhmsa.spec.datum.analysis.Analysis0D attribute), 52

datum\_dimensions (pyhmsa.spec.datum.analysis.Analysis1D attribute), 53

datum\_dimensions (pyhmsa.spec.datum.analysis.Analysis2D attribute), 53

datum\_dimensions (pyhmsa.spec.datum.analysislist.AnalysisList0D attribute), 53

datum\_dimensions (pyhmsa.spec.datum.analysislist.AnalysisList1D attribute), 53

datum\_dimensions (pyhmsa.spec.datum.analysislist.AnalysisList2D attribute), 54

## E

ElementalID (class  
pyhmsa.spec.condition.elementalid), 41

ElementalIDXray (class  
pyhmsa.spec.condition.elementalid), 41

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), 39

emission\_current (pyhmsa.spec.condition.probe.ProbeEM  
attribute), 44

emission\_current (pyhmsa.spec.condition.probe.ProbeTEM  
attribute), 46

end\_channel (pyhmsa.spec.condition.region.RegionOfInterest  
attribute), 48

energy (pyhmsa.spec.condition.elementalid.ElementalIDXray  
attribute), 42

exposure\_time (pyhmsa.spec.condition.detector.DetectorCamera  
attribute), 28

extractor\_bias (pyhmsa.spec.condition.probe.ProbeEM  
attribute), 44

extractor\_bias (pyhmsa.spec.condition.probe.ProbeTEM  
attribute), 46

## F

filament\_current (pyhmsa.spec.condition.probe.ProbeEM  
attribute), 44

filament\_current (pyhmsa.spec.condition.probe.ProbeTEM  
attribute), 46

filepath (pyhmsa.datafile.DataFile attribute), 55

focal\_length (pyhmsa.spec.condition.detector.DetectorCamera  
attribute), 28

formula (pyhmsa.spec.condition.specimen.Specimen at-  
tribute), 49

formula (pyhmsa.spec.condition.specimen.SpecimenLayer  
attribute), 48

formula (pyhmsa.spec.condition.specimen.SpecimenMultilayer  
attribute), 50

frame\_count (pyhmsa.spec.condition.acquisition.AcquisitionRasterImage  
attribute), 16

frame\_count (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY  
attribute), 18

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

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

## G

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

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

generate\_uid() (in module pyhmsa.type.uid), 57

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

get\_area() (pyhmsa.spec.condition.detector.DetectorCamera  
method), 28

get\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometer  
method), 31

get\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL  
method), 33

get\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS  
method), 36

get\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS  
method), 39

get\_atomic\_number() (pyhmsa.spec.condition.elementalid.ElementalID  
method), 41

get\_atomic\_number() (pyhmsa.spec.condition.elementalid.ElementalIDXray  
method), 42

get\_azimuth() (pyhmsa.spec.condition.detector.DetectorCamera  
method), 28

get\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometer  
method), 31

get\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL  
method), 33

get\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS  
method), 36

get\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS  
method), 39

get\_base\_level() (pyhmsa.spec.condition.detector.PulseHeightAnalyser  
method), 27

get\_beam\_current() (pyhmsa.spec.condition.probe.ProbeEM  
method), 44

get\_beam\_current() (pyhmsa.spec.condition.probe.ProbeTEM  
method), 46

get\_beam\_diameter() (pyhmsa.spec.condition.probe.ProbeEM  
method), 44

get\_beam\_diameter() (pyhmsa.spec.condition.probe.ProbeTEM  
method), 46

get\_beam\_voltage() (pyhmsa.spec.condition.probe.ProbeEM  
method), 44

get\_beam\_voltage() (pyhmsa.spec.condition.probe.ProbeTEM  
method), 46

get\_bias() (pyhmsa.spec.condition.detector.PulseHeightAnalyser  
method), 27

get\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometer  
method), 31

get\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL  
method), 33

get\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS  
method), 36

---

```

get_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS, 36
    method), 39
get_camera_magnification()
    (pyhmsa.spec.condition.probe.ProbeTEM
    method), 46
get_chamber_pressure() (pyhmsa.spec.condition.probe.ProbeEM
    method), 44
get_chamber_pressure() (pyhmsa.spec.condition.probe.ProbeTEM
    method), 46
get_channel_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
    method), 31
get_channel_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
    method), 33
get_channel_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
    method), 36
get_channel_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
    method), 39
get_channels() (pyhmsa.spec.condition.region.RegionOfInterest
    method), 48
get_coefficients() (pyhmsa.spec.condition.calibration.CalibrationPolynomial
    method), 25
get_collection_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
    method), 31
get_collection_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
    method), 33
get_collection_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
    method), 36
get_collection_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
    method), 39
get_composition() (pyhmsa.spec.condition.specimen.SpecimenMultilayer
    method), 49
get_composition() (pyhmsa.spec.condition.specimen.SpecimenMultilayer
    method), 48
get_composition() (pyhmsa.spec.condition.specimen.SpecimenMultilayer
    method), 50
get_convergence_angle()
    (pyhmsa.spec.condition.probe.ProbeTEM
    method), 46
get_crystal_2d() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
    method), 36
get_description() (pyhmsa.spec.condition.specimen.Specimen
    method), 49
get_description() (pyhmsa.spec.condition.specimen.SpecimenMultilayer
    method), 50
get_dispersion_element()
    (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
    method), 36
get_distance() (pyhmsa.spec.condition.detector.DetectorCamera
    method), 28
get_distance() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
    method), 31
get_distance() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL
    method), 33
get_distance() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
    method), 36
get_elevation() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS
    method), 36
get_elevation() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS
    method), 39
get_emission_current() (pyhmsa.spec.condition.probe.ProbeEM
    method), 45
get_emission_current() (pyhmsa.spec.condition.probe.ProbeTEM
    method), 46
get_end_channel() (pyhmsa.spec.condition.region.RegionOfInterest
    method), 48
get_energy() (pyhmsa.spec.condition.elementalid.ElementalIDXray
    method), 42
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), 46
get_filament_current() (pyhmsa.spec.condition.probe.ProbeEM
    method), 45
get_filament_current() (pyhmsa.spec.condition.probe.ProbeTEM
    method), 46
get_filter_length() (pyhmsa.spec.condition.detector.DetectorCamera
    method), 28
get_filter_length() (pyhmsa.spec.condition.specimen.Specimen
    method), 49

```



method), 49

get\_formula() (pyhmsa.spec.condition.specimen.SpecimenLayer method), 48

get\_formula() (pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51

get\_frame\_count() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 16

get\_frame\_count() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18

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.DetectorSpectrometer method), 33

get\_gun\_pressure() (pyhmsa.spec.condition.probe.ProbeEM method), 45

get\_gun\_pressure() (pyhmsa.spec.condition.probe.ProbeTEM method), 46

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\_lens\_mode() (pyhmsa.spec.condition.probe.ProbeTEM method), 47

get\_line() (pyhmsa.spec.condition.elementalid.ElementalIDXray method), 42

get\_magnification() (pyhmsa.spec.condition.detector.DetectorCamera method), 28

get\_manufacturer() (pyhmsa.spec.condition.detector.DetectorCamera method), 28

get\_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31

get\_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 33

get\_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 36

get\_manufacturer() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 39

get\_manufacturer() (pyhmsa.spec.condition.instrument.Instrument method), 43

get\_manufacturer() (pyhmsa.spec.condition.detector.WindowLayer method), 26

get\_nominal\_throughput() (pyhmsa.spec.condition.detector.DetectorCamera method), 28

get\_offset() (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24

get\_origin() (pyhmsa.spec.condition.specimen.Specimen method), 49

get\_origin() (pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51

get\_pixel\_count\_u() (pyhmsa.spec.condition.detector.DetectorCamera method), 28

get\_pixel\_count\_v() (pyhmsa.spec.condition.detector.DetectorCamera method), 29

get\_position() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 14

get\_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18

get\_position() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20

---

get_position_end() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLineCap method), 16	get_position_end() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31
get_position_start() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLineCap method), 16	get_position_start() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 34
get_positions() (pyhmsa.spec.condition.acquisition.AcquisitionMultiPoint method), 15	get_positions() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 37
get_positions() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLine method), 16	get_positions() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 40
get_positions() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18	get_positions() (pyhmsa.spec.condition.instrument.Instrument method), 43
get_positions() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20	get_positions() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
get_pulse_height_analyser() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 31	get_pulse_height_analyser() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34
get_quantity() (pyhmsa.spec.condition.calibration.CalibrationConstant method), 23	get_quantity() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37
get_quantity() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25	get_quantity() (pyhmsa.spec.condition.detector.DetectorSpectrometerXE method), 40
get_quantity() (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24	get_quantity() (pyhmsa.spec.condition.detector.DetectorCamera method), 29
get_quantity() (pyhmsa.spec.condition.calibration.CalibrationPolynomial method), 25	get_quantity() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31
get_r() (pyhmsa.spec.condition.specimenposition.SpecimenPosition method), 51	get_r() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34
get_raster_mode() (pyhmsa.spec.condition.acquisition.AcquisitionRasterLine method), 16	get_raster_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37
get_raster_mode() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18	get_raster_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerXE method), 40
get_raster_mode() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20	get_raster_mode() (pyhmsa.spec.condition.region.RegionOfInterest method), 28
get_raster_mode_z() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 20	get_raster_mode_z() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 16
get_rowland_circle_diameter() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37	get_rowland_circle_diameter() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18
get_scan_magnification() (pyhmsa.spec.condition.probe.ProbeEM method), 45	get_scan_magnification() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 20
get_scan_magnification() (pyhmsa.spec.condition.probe.ProbeTEM method), 47	get_scan_magnification() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 18
get_semi_angle() (pyhmsa.spec.condition.detector.DetectorCamera method), 29	get_semi_angle() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY method), 20
get_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31	get_semi_angle() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 16
get_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34	get_semi_angle() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 18
get_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37	get_semi_angle() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20
get_semi_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometerXE method), 40	get_semi_angle() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 18
get_serial_number() (pyhmsa.spec.condition.detector.DetectorCamera method), 29	get_semi_angle() (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 20

[get\\_step\\_size\\_z\(\)](#) (pyhmsa.spec.condition.acquisition.AcquisitionPointXYZ method), 20  
[get\\_strobe\\_rate\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 40  
[get\\_symbol\(\)](#) (pyhmsa.spec.condition.elementalid.ElementalID method), 41  
[get\\_symbol\(\)](#) (pyhmsa.spec.condition.elementalid.ElementalID method), 42  
[get\\_t\(\)](#) (pyhmsa.spec.condition.specimenposition.SpecimenPosition method), 52  
[get\\_technology\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 40  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 29  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 31  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 34  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 40  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.specimen.SpecimenLayer method), 49  
[get\\_temperature\(\)](#) (pyhmsa.spec.condition.specimen.SpecimenLayer method), 51  
[get\\_thickness\(\)](#) (pyhmsa.spec.condition.detector.WindowLayer method), 26  
[get\\_thickness\(\)](#) (pyhmsa.spec.condition.specimen.SpecimenLayer method), 48  
[get\\_time\\_constant\(\)](#) (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 40  
[get\\_total\\_time\(\)](#) (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint method), 15  
[get\\_total\\_time\(\)](#) (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 14  
[get\\_total\\_time\(\)](#) (pyhmsa.spec.condition.acquisition.AcquisitionRaster2D method), 16  
[get\\_total\\_time\(\)](#) (pyhmsa.spec.condition.acquisition.AcquisitionRaster2D method), 18  
[get\\_total\\_time\(\)](#) (pyhmsa.spec.condition.acquisition.AcquisitionRaster2D method), 20  
[get\\_unit\(\)](#) (pyhmsa.spec.condition.calibration.CalibrationConstant method), 23  
[get\\_unit\(\)](#) (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25  
[get\\_unit\(\)](#) (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24  
[get\\_unit\(\)](#) (pyhmsa.spec.condition.calibration.CalibrationPolynomial method), 25  
[get\\_unit\(\)](#) (pyhmsa.spec.condition.composition.CompositionElemental method), 22  
[get\\_value\(\)](#) (pyhmsa.spec.condition.calibration.CalibrationConstant method), 23

## H

[Header](#) (class in pyhmsa.spec.header), 13  
[header](#) (pyhmsa.datafile.DataFile attribute), 55  
[I](#)  
[ImageRaster2D](#) (class in pyhmsa.spec.datum.imageraster), 54  
[ImageRaster2DHyperimage](#) (class in pyhmsa.spec.datum.imageraster), 54  
[ImageRaster2DSpectral](#) (class in pyhmsa.spec.datum.imageraster), 54  
[Instrument](#) (class in pyhmsa.spec.condition.instrument), 42  
[is\\_bulk\(\)](#) (pyhmsa.spec.condition.specimen.SpecimenLayer method), 48  
[items\(\)](#) (pyhmsa.spec.condition.composition.CompositionElemental method), 22  
[iterkeys\(\)](#) (pyhmsa.spec.condition.composition.CompositionElemental method), 22  
[itervalues\(\)](#) (pyhmsa.spec.condition.composition.CompositionElemental method), 22



## K

keys() (pyhmsa.spec.condition.composition.CompositionElemental method), 22

## L

labels (pyhmsa.spec.condition.calibration.CalibrationExplicit attribute), 25

langstr (class in pyhmsa.type.language), 57

language (pyhmsa.datafile.DataFile attribute), 55

layers (pyhmsa.spec.condition.detector.Window attribute), 26

layers (pyhmsa.spec.condition.specimen.SpecimenMultilayer attribute), 51

lens\_mode (pyhmsa.spec.condition.probe.ProbeTEM attribute), 47

line (pyhmsa.spec.condition.elementalid.ElementalIDXray attribute), 42

## M

magnification (pyhmsa.spec.condition.detector.DetectorCamera attribute), 29

manufacturer (pyhmsa.spec.condition.detector.DetectorCamera attribute), 29

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

manufacturer (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 34

manufacturer (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 37

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

manufacturer (pyhmsa.spec.condition.instrument.Instrument attribute), 43

material (pyhmsa.spec.condition.detector.WindowLayer attribute), 26

measurement\_unit (pyhmsa.spec.condition.detector.DetectorCamera attribute), 29

measurement\_unit (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 31

measurement\_unit (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 34

measurement\_unit (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 37

measurement\_unit (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 40

merge() (pyhmsa.datafile.DataFile method), 55

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

model (pyhmsa.spec.condition.detector.DetectorCamera attribute), 29

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

model (pyhmsa.spec.condition.detector.DetectorSpectrometerCL attribute), 34

model (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS attribute), 37

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

model (pyhmsa.spec.condition.instrument.Instrument attribute), 43

## N

name (pyhmsa.spec.condition.specimen.Specimen attribute), 49

name (pyhmsa.spec.condition.specimen.SpecimenLayer attribute), 48

name (pyhmsa.spec.condition.specimen.SpecimenMultilayer attribute), 51

nominal\_throughput (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 40

notation (pyhmsa.type.xrayline.xrayline attribute), 58

## O

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

origin (pyhmsa.spec.condition.specimen.Specimen attribute), 49

origin (pyhmsa.spec.condition.specimen.SpecimenMultilayer attribute), 51

operator\_conditions (pyhmsa.datafile.DataFile attribute), 55

operator (pyhmsa.spec.header.Header attribute), 13

## P

parse\_unit() (in module pyhmsa.type.unit), 57

pixel\_count\_u (pyhmsa.spec.condition.detector.DetectorCamera attribute), 29

pixel\_count\_v (pyhmsa.spec.condition.detector.DetectorCamera attribute), 29

pop() (pyhmsa.spec.condition.composition.CompositionElemental method), 22

popitem() (pyhmsa.spec.condition.composition.CompositionElemental method), 22

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

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

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

position\_end (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

position\_start (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 16

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

positions (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 17

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

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

ProbeEM (class in pyhmsa.spec.condition.probe), 44

ProbeTEM (class in pyhmsa.spec.condition.probe), 45

pulse\_height\_analyser (pyhmsa.spec.condition.detector.DetectorSpecimenWDS attribute), 37

PulseHeightAnalyser (class in pyhmsa.spec.condition.probe.GUN\_TYPE\_LAB6 pyhmsa.spec.condition.detector), 26

pyhmsa.spec.condition.acquisition.POSITION\_LOCATION\_CENTER (built-in variable), 14

pyhmsa.spec.condition.acquisition.POSITION\_LOCATION\_END (built-in variable), 14

pyhmsa.spec.condition.acquisition.POSITION\_LOCATION\_START (built-in variable), 14

pyhmsa.spec.condition.acquisition.RASTER\_MODE\_BEAM (built-in variable), 14

pyhmsa.spec.condition.acquisition.RASTER\_MODE\_STAGE (built-in variable), 14

pyhmsa.spec.condition.acquisition.RASTER\_MODE\_Z\_FIB (built-in variable), 14

pyhmsa.spec.condition.detector.COLLECTION\_MODE\_PAP (built-in variable), 23

pyhmsa.spec.condition.detector.COLLECTION\_MODE\_SERIAL (built-in variable), 23

pyhmsa.spec.condition.detector.PHA\_MODE\_DIFFERENTIAL (built-in variable), 23

pyhmsa.spec.condition.detector.PHA\_MODE\_INTEGRAL (built-in variable), 23

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_AES (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_BEI (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_CLS (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_EBSD (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_EDS (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_ELS (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_GAM (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_PES (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_SEI (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_WDS (built-in variable), 22

pyhmsa.spec.condition.detector.SIGNAL\_TYPE\_XRF (built-in variable), 22

pyhmsa.spec.condition.detector.XEDS\_TECHNOLOGY\_GE (built-in variable), 23

pyhmsa.spec.condition.detector.XEDS\_TECHNOLOGY\_SDD (built-in variable), 23

pyhmsa.spec.condition.detector.XEDS\_TECHNOLOGY\_SILI (built-in variable), 23

pyhmsa.spec.condition.detector.XEDS\_TECHNOLOGY\_UCAL (built-in variable), 23

pyhmsa.spec.condition.probe.GUN\_TYPE\_COLD\_FEG (built-in variable), 43

pyhmsa.spec.condition.probe.GUN\_TYPE\_LAB6 (built-in variable), 43

pyhmsa.spec.condition.probe.GUN\_TYPE\_SCHOTTKY\_FEG (built-in variable), 43

pyhmsa.spec.condition.probe.GUN\_TYPE\_W\_FILAMENT (built-in variable), 43

pyhmsa.spec.condition.probe.LENS\_MODE\_DIFFR (built-in variable), 43

pyhmsa.spec.condition.probe.LENS\_MODE\_IMAGE (built-in variable), 43

pyhmsa.spec.condition.probe.LENS\_MODE\_SCDIF (built-in variable), 43

pyhmsa.spec.condition.probe.LENS\_MODE\_SCIMG (built-in variable), 43

pyhmsa.type.checksum.CHECKSUM\_ALGORITHM\_SHA1 (built-in variable), 56

pyhmsa.type.checksum.CHECKSUM\_ALGORITHM\_SUM32 (built-in variable), 56

pyhmsa.type.xrayline.NOTATION\_IUPAC (built-in variable), 58

pyhmsa.type.xrayline.NOTATION\_SIEGBAHN (built-in variable), 58

## Q

quantity (pyhmsa.spec.condition.calibration.CalibrationConstant attribute), 23

quantity (pyhmsa.spec.condition.calibration.CalibrationExplicit attribute), 25

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

quantity (pyhmsa.spec.condition.calibration.CalibrationPolynomial attribute), 25

## R

r (pyhmsa.spec.condition.specimenposition.SpecimenPosition attribute), 52

raster\_mode (pyhmsa.spec.condition.acquisition.AcquisitionRasterLinescan attribute), 17

raster\_mode (pyhmsa.spec.condition.acquisition.AcquisitionRasterXY attribute), 18

raster\_mode (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

raster\_mode\_z (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ attribute), 20

read() (pyhmsa.datafile.DataFile class method), 55

RegionOfInterest (class in set\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37  
 pyhmsa.spec.condition.region), 47  
 rowland\_circle\_diameter set\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40  
 (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 40  
 attribute), 37  
 set\_base\_level() (pyhmsa.spec.condition.detector.PulseHeightAnalyser method), 27  
 set\_beam\_current() (pyhmsa.spec.condition.probe.ProbeEM method), 45  
 set\_beam\_current() (pyhmsa.spec.condition.probe.ProbeTEM method), 47  
 set\_beam\_diameter() (pyhmsa.spec.condition.probe.ProbeEM method), 45  
 set\_beam\_diameter() (pyhmsa.spec.condition.probe.ProbeTEM method), 47  
 set\_beam\_voltage() (pyhmsa.spec.condition.probe.ProbeEM method), 45  
 set\_beam\_voltage() (pyhmsa.spec.condition.probe.ProbeTEM method), 47  
 set\_bias() (pyhmsa.spec.condition.detector.PulseHeightAnalyser method), 27  
 set\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31  
 set\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34  
 set\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37  
 set\_calibration() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40  
 set\_camera\_magnification() (pyhmsa.spec.condition.probe.ProbeTEM method), 47  
 set\_chamber\_pressure() (pyhmsa.spec.condition.probe.ProbeEM method), 45  
 set\_chamber\_pressure() (pyhmsa.spec.condition.probe.ProbeTEM method), 47  
 set\_channel\_count() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31  
 set\_channel\_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34  
 set\_channel\_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37  
 set\_channel\_count() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40  
 set\_channels() (pyhmsa.spec.condition.region.RegionOfInterest method), 48  
 set\_coefficients() (pyhmsa.spec.condition.calibration.CalibrationPolynomial method), 25  
 set\_collection\_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32  
 set\_collection\_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34  
 set\_collection\_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37  
 set\_collection\_mode() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40  
 set\_area() (pyhmsa.spec.condition.detector.DetectorCamera method), 29  
 set\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31  
 set\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34  
 set\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37  
 set\_area() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40  
 set\_atomic\_number() (pyhmsa.spec.condition.elementalid.ElementalID method), 41  
 set\_atomic\_number() (pyhmsa.spec.condition.elementalid.ElementalID method), 42  
 set\_azimuth() (pyhmsa.spec.condition.detector.DetectorCamera method), 29  
 set\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 31  
 set\_azimuth() (pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34

```

method), 40
set_composition()(pyhmsa.spec.condition.specimen.Specimen method), 37
method), 49
set_composition()(pyhmsa.spec.condition.specimen.SpecimenLayer method), 40
method), 49
set_composition()(pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 45
method), 51
set_convergence_angle()(pyhmsa.spec.condition.probe.ProbeTEM method), 47
method), 47
set_crystal_2d()(pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 42
method), 37
set_description()(pyhmsa.spec.condition.specimen.Specimen method), 29
method), 50
set_description()(pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 45
method), 51
set_dispersion_element()
(pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37
set_distance()(pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_distance()(pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 32
set_distance()(pyhmsa.spec.condition.detector.DetectorSpectrometerEDS method), 34
set_distance()(pyhmsa.spec.condition.detector.DetectorSpectrometerWDS method), 37
set_distance()(pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40
set_dwelling_time()(pyhmsa.spec.condition.acquisition.AcquisitionMultipoint method), 15
set_dwelling_time()(pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 14
set_dwelling_time()(pyhmsa.spec.condition.acquisition.AcquisitionRange method), 17
set_dwelling_time()(pyhmsa.spec.condition.acquisition.AcquisitionRangeXY method), 18
set_dwelling_time()(pyhmsa.spec.condition.acquisition.AcquisitionRangeXYZ method), 20
set_dwelling_time_live()(pyhmsa.spec.condition.acquisition.AcquisitionMeasurement method), 15
set_dwelling_time_live()(pyhmsa.spec.condition.acquisition.AcquisitionPulse method), 14
set_dwelling_time_live()(pyhmsa.spec.condition.acquisition.AcquisitionRasterLine method), 17
set_dwelling_time_live()(pyhmsa.spec.condition.acquisition.AcquisitionRasterLine method), 17
set_dwelling_time_live()(pyhmsa.spec.condition.acquisition.AcquisitionRasterLine method), 18
set_dwelling_time_live()(pyhmsa.spec.condition.acquisition.AcquisitionRasterLine method), 18
set_elevation()(pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_elevation()(pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32
set_elevation()(pyhmsa.spec.condition.detector.DetectorSpectrometer method), 34
set_elevation()(pyhmsa.spec.condition.elementalid.ElementalIDXray method), 42
set_emission_current()(pyhmsa.spec.condition.probe.ProbeEM method), 45
set_emission_current()(pyhmsa.spec.condition.probe.ProbeTEM method), 45
set_energy()(pyhmsa.spec.condition.elementalid.ElementalIDXray method), 42
set_exposure_time()(pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_extractor_bias()(pyhmsa.spec.condition.probe.ProbeEM method), 45
set_extractor_bias()(pyhmsa.spec.condition.probe.ProbeTEM method), 47
set_filament_current()(pyhmsa.spec.condition.probe.ProbeTEM method), 47
set_focal_length()(pyhmsa.spec.condition.detector.DetectorCamera method), 29
set_function()(pyhmsa.spec.condition.specimen.Specimen method), 50
set_function()(pyhmsa.spec.condition.specimen.SpecimenLayer method), 49
set_function()(pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51
set_image_size()(pyhmsa.spec.condition.calibration.CalibrationLinear method), 24
set_image_size()(pyhmsa.spec.condition.detector.PulseHeightAnalyser method), 27
set_image_size()(pyhmsa.spec.condition.detector.DetectorSpectrometerCL method), 34
set_image_size()(pyhmsa.spec.condition.probe.ProbeEM method), 45
set_image_size()(pyhmsa.spec.condition.probe.ProbeTEM method), 47
set_image_size()(pyhmsa.spec.condition.probe.ProbeTEM method), 47
set_image_size()(pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25
set_lens_mode()(pyhmsa.spec.condition.probe.ProbeTEM method), 47
set_line()(pyhmsa.spec.condition.elementalid.ElementalIDXray method), 42
set_magnification()(pyhmsa.spec.condition.detector.DetectorCamera method), 29

```

73



method), 29

set\_serial\_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 32

set\_serial\_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 34

set\_serial\_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 37

set\_serial\_number() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40

set\_serial\_number() (pyhmsa.spec.condition.instrument.Instrument method), 43

set\_signal\_type() (pyhmsa.spec.condition.detector.DetectorCamera method), 29

set\_signal\_type() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 32

set\_signal\_type() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 34

set\_signal\_type() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 38

set\_signal\_type() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 40

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.DetectorSpectrometer method), 34

set\_solid\_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 38

set\_solid\_angle() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 40

set\_step\_count() (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint method), 17

set\_step\_count\_x() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 19

set\_step\_count\_x() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 21

set\_step\_count\_y() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 19

set\_step\_count\_y() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 21

set\_step\_count\_z() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 21

set\_step\_size() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 17

set\_step\_size\_x() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 19

set\_step\_size\_x() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 21

set\_step\_size\_y() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 19

set\_step\_size\_y() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 21

set\_step\_size\_z() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 21

method), 21

set\_spectrometer() (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS method), 40

set\_symbolic() (pyhmsa.spec.condition.elementalid.ElementalID method), 41

set\_symbolic() (pyhmsa.spec.condition.elementalid.ElementalID method), 42

set\_symbolic() (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.DetectorSpectrometer method), 34

set\_temperature() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 38

set\_temperature() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 41

set\_temperature() (pyhmsa.spec.condition.specimen.Specimen method), 50

set\_temperature() (pyhmsa.spec.condition.specimen.SpecimenMultilayer method), 51

set\_thickness() (pyhmsa.spec.condition.detector.WindowLayer method), 26

set\_thickness() (pyhmsa.spec.condition.specimen.SpecimenLayer method), 49

set\_thickness() (pyhmsa.spec.condition.detector.DetectorSpectrometer method), 41

set\_ratio() (pyhmsa.spec.condition.acquisition.AcquisitionMultipoint method), 15

set\_ratio() (pyhmsa.spec.condition.acquisition.AcquisitionPoint method), 15

set\_ratio() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 17

set\_ratio() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 19

set\_ratio() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 21

set\_ratio() (pyhmsa.spec.condition.acquisition.AcquisitionRaster method), 23

set\_ratio() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 25

set\_ratio() (pyhmsa.spec.condition.calibration.CalibrationLinear method), 24

set\_ratio() (pyhmsa.spec.condition.calibration.CalibrationPolynomial method), 25

set\_ratio() (pyhmsa.spec.condition.composition.CompositionElemental method), 22

set\_ratio() (pyhmsa.spec.condition.calibration.CalibrationConstant method), 23

set\_ratio() (pyhmsa.spec.condition.calibration.CalibrationExplicit method), 23



TEMPLATE (pyhmsa.spec.condition.acquisition.AcquisitionRasterXYZ method), 54  
     attribute), 20  
 TEMPLATE (pyhmsa.spec.condition.composition.CompositionElemental attribute), 15  
     attribute), 21  
 TEMPLATE (pyhmsa.spec.condition.detector.DetectorCamera attribute), 15  
     attribute), 28  
 TEMPLATE (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 17  
     attribute), 30  
 TEMPLATE (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 19  
     attribute), 33  
 TEMPLATE (pyhmsa.spec.condition.detector.DetectorSpectrometer attribute), 21  
     attribute), 36  
 TEMPLATE (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 39  
 TEMPLATE (pyhmsa.spec.condition.elementalid.ElementalID (pyhmsa.spec.datum.analysislist.AnalysisList2D attribute), 53  
     attribute), 41  
 TEMPLATE (pyhmsa.spec.condition.elementalid.ElementalIDXray (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage attribute), 54  
     attribute), 42  
 TEMPLATE (pyhmsa.spec.condition.instrument.Instrument unit (pyhmsa.spec.condition.calibration.CalibrationConstant attribute), 23  
     attribute), 43  
 TEMPLATE (pyhmsa.spec.condition.probe.ProbeEM attribute), 44  
     attribute), 44  
 TEMPLATE (pyhmsa.spec.condition.probe.ProbeTEM attribute), 46  
     attribute), 46  
 TEMPLATE (pyhmsa.spec.condition.region.RegionOfInterest unit (pyhmsa.spec.condition.calibration.CalibrationExplicit attribute), 25  
     attribute), 48  
 TEMPLATE (pyhmsa.spec.condition.specimen.Specimen unit (pyhmsa.spec.condition.calibration.CalibrationLinear attribute), 24  
     attribute), 49  
 TEMPLATE (pyhmsa.spec.condition.specimen.SpecimenMultilayer unit (pyhmsa.spec.condition.calibration.CalibrationPolynomial attribute), 25  
     attribute), 50  
 TEMPLATE (pyhmsa.spec.condition.specimenposition.SpecimenPosition unit (pyhmsa.spec.condition.composition.CompositionElemental attribute), 22  
     attribute), 51  
 thickness (pyhmsa.spec.condition.detector.WindowLayer attribute), 26  
 thickness (pyhmsa.spec.condition.specimen.SpecimenLayer attribute), 49  
 time (pyhmsa.spec.header.Header attribute), 13  
 time\_constant (pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS attribute), 41  
 timezone (pyhmsa.spec.header.Header attribute), 13  
 title (pyhmsa.spec.header.Header attribute), 13  
 to\_wt() (pyhmsa.spec.condition.composition.CompositionElemental method), 22  
 toanalysis() (pyhmsa.spec.datum.analysislist.AnalysisList0D validate\_dtype() (in module pyhmsa.type.numerical), 57  
     method), 53  
 toanalysis() (pyhmsa.spec.datum.analysislist.AnalysisList1D validate\_identifier() (in module pyhmsa.type.identifier), 56  
     method), 53  
 toanalysis() (pyhmsa.spec.datum.analysislist.AnalysisList2D validate\_language\_tag() (in module pyhmsa.type.language), 56  
     method), 54  
 toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2D validate\_unit() (in module pyhmsa.type.unit), 57  
     method), 54  
 toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage validate\_value() (pyhmsa.spec.condition.calibration.CalibrationConstant attribute), 23  
     method), 54  
 toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage values (pyhmsa.spec.condition.calibration.CalibrationExplicit attribute), 25  
     method), 54  
 toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DHyperimage values() (pyhmsa.spec.condition.composition.CompositionElemental method), 22  
     method), 54  
 toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral version (pyhmsa.datafile.DataFile attribute), 55  
     method), 54  
 toanalysis() (pyhmsa.spec.datum.imageraster.ImageRaster2DSpectral version (pyhmsa.datafile.DataFile attribute), 55



## W

Window (class in `pyhmsa.spec.condition.detector`), [26](#)  
window (`pyhmsa.spec.condition.detector.DetectorSpectrometerWDS`  
attribute), [38](#)  
window (`pyhmsa.spec.condition.detector.DetectorSpectrometerXEDS`  
attribute), [41](#)  
window (`pyhmsa.spec.condition.detector.PulseHeightAnalyser`  
attribute), [27](#)  
WindowLayer (class in `pyhmsa.spec.condition.detector`),  
[26](#)  
working\_distance (`pyhmsa.spec.condition.probe.ProbeEM`  
attribute), [45](#)  
working\_distance (`pyhmsa.spec.condition.probe.ProbeTEM`  
attribute), [47](#)  
write() (`pyhmsa.datafile.DataFile` method), [55](#)

## X

x (`pyhmsa.spec.condition.specimenposition.SpecimenPosition`  
attribute), [52](#)  
xrayline (class in `pyhmsa.type.xrayline`), [58](#)

## Y

y (`pyhmsa.spec.condition.specimenposition.SpecimenPosition`  
attribute), [52](#)

## Z

z (`pyhmsa.spec.condition.specimenposition.SpecimenPosition`  
attribute), [52](#)