

---

# **afmformats Documentation**

***Release 0.14.3***

**Paul Müller**

**Jul 01, 2021**



# CONTENTS

<b>1</b>	<b>Getting started</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Basic Usage . . . . .	3
1.3	Supported file formats . . . . .	3
1.4	Notes . . . . .	4
<b>2</b>	<b>Advanced Usage</b>	<b>5</b>
2.1	Grouping AFM data . . . . .	5
<b>3</b>	<b>afmformats</b>	<b>7</b>
3.1	Submodules . . . . .	7
3.1.1	afmformats._version . . . . .	7
3.1.2	afmformats._version_save . . . . .	8
3.1.3	afmformats.afm_data . . . . .	8
3.1.4	afmformats.afm_group . . . . .	10
3.1.5	afmformats.afm_qmap . . . . .	11
3.1.6	afmformats.errors . . . . .	13
3.1.7	afmformats.fmt_hdf5 . . . . .	15
3.1.8	afmformats.fmt_igor . . . . .	16
3.1.9	afmformats.fmt_jpk . . . . .	17
3.1.10	afmformats.fmt_ntmdt_txt . . . . .	24
3.1.11	afmformats.fmt_tab . . . . .	24
3.1.12	afmformats.fmt_workshop . . . . .	25
3.1.13	afmformats.formats . . . . .	27
3.1.14	afmformats.lazy_loader . . . . .	30
3.1.15	afmformats.meta . . . . .	31
3.1.16	afmformats.mod_force_distance . . . . .	36
3.1.17	afmformats.parse_funcs . . . . .	37
<b>4</b>	<b>Changelog</b>	<b>39</b>
4.1	version 0.14.3 . . . . .	39
4.2	version 0.14.2 . . . . .	39
4.3	version 0.14.1 . . . . .	39
4.4	version 0.14.0 . . . . .	40
4.5	version 0.13.3 . . . . .	40
4.6	version 0.13.2 . . . . .	40
4.7	version 0.13.1 . . . . .	40
4.8	version 0.13.0 . . . . .	40
4.9	version 0.12.6 . . . . .	41
4.10	version 0.12.5 . . . . .	41

4.11	version 0.12.4	41
4.12	version 0.12.3	41
4.13	version 0.12.2	41
4.14	version 0.12.1	41
4.15	version 0.12.0	41
4.16	version 0.11.0	42
4.17	version 0.10.2	42
4.18	version 0.10.1	42
4.19	version 0.10.0	42
4.20	version 0.9.0	42
4.21	version 0.8.0	42
4.22	version 0.7.1	43
4.23	version 0.7.0	43
4.24	version 0.6.0	43
4.25	version 0.5.2	43
4.26	version 0.5.1	43
4.27	version 0.5.0	43
4.28	version 0.4.1	44
4.29	version 0.4.0	44
4.30	version 0.3.0	44
4.31	version 0.2.0	44
4.32	version 0.1.0	44
<b>5</b>	<b>Bibliography</b>	<b>45</b>
<b>6</b>	<b>Indices and tables</b>	<b>47</b>
	<b>Bibliography</b>	<b>49</b>
	<b>Python Module Index</b>	<b>51</b>
	<b>Index</b>	<b>53</b>

Afmformats is a Python library for reading atomic force microscopy (AFM) data file formats. This is the documentation of afmformats version 0.14.3.



## GETTING STARTED

### 1.1 Installation

To install `afmformats`, use one of the following methods (the package dependencies will be installed automatically):

- from **PyPI**: `pip install afmformats`
- from **sources**: `pip install -e .`

### 1.2 Basic Usage

```
In [1]: import afmformats

In [2]: dslist = afmformats.load_data("data/force-save-example.jpik-force")

# dslist is a list of force-distance curves
In [3]: dslist
Out[3]: [<AFMForceDistance 'data/force-save-example.jpik-force'[0] at 0x7f5d4563eba8>]

# available data columns of the first curve
In [4]: dslist[0].columns
Out[4]: ['force', 'height (measured)', 'height (piezo)', 'segment', 'time']

In [5]: dslist[0]["force"]
Out[5]:
array([-6.56678981e-10, -6.64172230e-10, -6.79510911e-10, ...,
       -7.61449435e-10, -7.68909858e-10, -7.58163174e-10])
```

### 1.3 Supported file formats

All supported file formats are listed in the table below. If you are interested in other file formats, please [create a new issue](#).

Format introduced by	Description	Extension	Loader
AFM workshop	comma-separated values	.csv	<code>fmt_workshop.ws_single.load_csv</code>
afmformats	HDF5 based	.h5	<code>fmt_hdf5.load_hdf5</code>
Asylum Research	binary-wave	.ibw	<code>fmt_igor.load_igor</code>
JPK Instruments	binary FD data	.jpk-force	<code>fmt_jpk.load_jpk</code>
JPK Instruments	binary QMap data	.jpk-force-map	<code>fmt_jpk.load_jpk</code>
JPK Instruments	binary QMap data	.jpk-qi-data	<code>fmt_jpk.load_jpk</code>
afmformats	tab-separated values	.tab	<code>fmt_tab.load_tab</code>
NT-MDT Spectrum Instruments	exported by NT-MDT Nova	.txt	<code>fmt_ntmdt_txt.load_txt</code>
AFM workshop	zipped zip comma-separated values	.zip	<code>fmt_workshop.ws_map.load_map</code>

## 1.4 Notes

Afmformats is a base module for loading experimental data. You might want to use [nanite](#) or [PyJibe](#) for higher-level functionalities.



## ADVANCED USAGE

### 2.1 Grouping AFM data

AFM data can be organized in an *afmformats.AFMGroup* which comes with a few user-convenient functionalities:

```
In [1]: import afmformats

In [2]: group = afmformats.AFMGroup("data/force-map2x2-example.jpj-force-map")

# group contains all curves in the data file
In [3]: print(group)
AFMGroup: 'data/force-map2x2-example.jpj-force-map'
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [0]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [1]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [2]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [3]

# you may add other data files to groups
In [4]: group += afmformats.load_data("data/force-save-example.jpj-force")

In [5]: print(group)
AFMGroup: 'None'
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [0]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [1]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [2]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [3]
- AFMForceDistance 'data/force-save-example.jpj-force' [0]

# You can also extract a subgroup that matches a certain path
In [6]: subgroup = group.subgroup_with_path("data/force-map2x2-example.jpj-force-map")

In [7]: print(subgroup)
AFMGroup: 'data/force-map2x2-example.jpj-force-map'
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [0]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [1]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [2]
- AFMForceDistance 'data/force-map2x2-example.jpj-force-map' [3]
```



- *Submodules*

## 3.1 Submodules

### 3.1.1 afmformats.\_version

Determine package version from git repository tag

Each time this file is imported it checks whether the package version can be determined using *git describe*. If this fails (because either this file is not located at the 1st level down the repository root or it is not under version control), the version is read from the script “\_version\_save.py” which is not versioned by git, but always included in the final distribution archive (e.g. via PyPI). If the git version does not match the saved version, then “\_version\_save.py” is updated.

#### Usage

1. Put this file in your main module directory:  
    REPO\_ROOT/package\_name/\_version.py
2. Add this line to REPO\_ROOT/package\_name/\_\_init\_\_.py  
    from .\_version import version as \_\_version\_\_ # noqa: F401
3. (Optional) Add this line to REPO\_ROOT/.gitignore  
    \_version\_save.py

#### Features

- supports Python 2 and 3
- supports frozen applications (e.g. PyInstaller)
- supports installing into a virtual environment that is located in a git repository
- saved version is located in a python file and therefore no other files (e.g. MANIFEST.in) need be edited
- fallback version is the creation date
- excluded from code coverage via “pragma: no cover”

## Changelog

### 2019-11-06.2

- use `os.path.split` instead of counting `os.path.sep` (Windows)

### 2019-11-06

- remove deprecated `imp` dependency (replace with `parser`)
- check whether this file is versioned and its location is correct
- code cleanup and docs update

## 3.1.2 afmformats.\_version\_save

## 3.1.3 afmformats.afm\_data

- *Classes*
- *Variables*

### Classes

- *AFMData*: General base class for AFM data

**class** afmformats.afm\_data.**AFMData**(*data*, *metadata*, *diskcache=False*)  
General base class for AFM data

Initialization

#### Parameters

- **data** (*dict-like*) – Experimental data
- **metadata** (*dict*) – Metadata
- **diskcache** (*bool*) – TODO

### Inheritance



**export\_data**(*out*, *metadata=True*, *fmt='tab'*)  
Export all data columns to a file

#### Parameters

- **out** (*str*, *pathlib.Path*, *writable io.IOBase*, or *h5py.Group*) – Output path, open file, or h5py object
- **metadata** (*bool* or *list*) – If True, all available metadata are stored. If False, no metadata are stored. If a list, only the given metadata keys are stored.
- **fmt** (*str*) – “tab” for the tab separated values format and “hdf5” / “h5” for the HDF5 file format

## Notes

- If you wish to append HDF5 data to an existing file, please open the file first and call this function with the h5py.File object, i.e.

```
with h5py.File(path, "a") as h5:
    fdist.export(out=h5, fmt="hdf5")
```

Otherwise the file will be overridden.

- The column “index” is not exported in the HDF5 file format

### property columns

Available data columns

### property enum

Unique index of *self* in *self.path*

Indexing starts at “0”

### property metadata

Unique index of *self* in *self.path*

### abstract property modality

Imaging modality (e.g. force-distance)

### property path

Path to the measurement file

## Variables

- *column\_dtypes*
- *column\_units*
- *known\_columns*

### afmformats.afm\_data.column\_dtypes

Data types of all known columns (all other columns are assumed to be float)

```
{'force': <class 'float'>,
 'height (measured)': <class 'float'>,
 'height (piezo)': <class 'float'>,
 'index': <class 'int'>,
 'segment': <class 'bool'>,
 'time': <class 'float'>,
 'tip position': <class 'float'>}
```

### afmformats.afm\_data.column\_units

Units of all known columns

```
{'force': 'N',
 'height (measured)': 'm',
 'height (piezo)': 'm',
 'index': '',
 'segment': '',
 'time': 's',
 'tip position': 'm'}
```

`afmformats.afm_data.known_columns`

Known data columns

```
['force',
 'height (measured)',
 'height (piezo)',
 'index',
 'segment',
 'time',
 'tip position']
```

### 3.1.4 afmformats.afm\_group

- *Classes*

#### Classes

- *AFMGroup*: Container for `afmformats.afm_data.AFMData`

```
class afmformats.afm_group.AFMGroup(path=None, meta_override=None, callback=None,
                                     data_classes_by_modality=None)
```

Container for `afmformats.afm_data.AFMData`

#### Parameters

- **path** (*str* or *pathlib.Path* or *None*) – If this option is specified, then an AFMGroup is generated directly from a datafile.
- **meta\_override** (*dict*) – Dictionary with metadata that is used when loading the data in *path*.
- **callback** (*callable* or *None*) – A method that accepts a float between 0 and 1 to externally track the process of loading the data.
- **data\_classes\_by\_modality** (*dict*) – Override the default AFMData class to use for managing the data (see `default_data_classes_by_modality`): This is e.g. used by `nanite:index` to pass *Indentation* (which is a subclass of the default *AFMForceDistance*) for handling “force-indentation” data.

## Inheritance

AFMGroup

**append**(*afmdata*)

Append an AFMData instance

**Parameters** *afmdata* (`afmformats.afm_data.AFMData`) – AFM data

**get\_enum**(*enum*)

Return the AFMData curve with this enum value

**Raises**

- **ValueError** if multiple curves with the same enum value exist. –
- **KeyError** if the enum value is not found –

**subgroup\_with\_path**(*path*)

Return a subgroup with AFMData matching *path*

### 3.1.5 afmformats.afm\_qmap

- *Functions*
- *Classes*
- *Variables*

## Functions

- *qmap\_feature*(): Decorator for labeling AFMQMap features

`afmformats.afm_qmap.qmap_feature(name, unit, cache=False)`

Decorator for labeling AFMQMap features

The name and unit are stored as properties of the wrapped function. In addition, the return value of the function can be cached (see *cache* argument).

**Parameters**

- **name** (*str*) – Name of the feature
- **unit** (*str*) – Unit of the returned feature
- **cache** (*bool* or *callable*) – If boolean, determines whether the feature data should be cached or not. If callable, the callable gets an instance of AFMData as an argument and should return an identifier (*str*) for the current value. If that identifier is the same as in the cache, then the cached value is used.

## Classes

- *AFMQMap*: Management of quantitative AFM data on a grid

```
class afmformats.afm_qmap.AFMQMap(path_or_group, meta_override=None, callback=None,
                                   data_classes_by_modality=None)
```

Management of quantitative AFM data on a grid

### Parameters

- **path\_or\_group** (*str* or *pathlib.Path* or *afmformats.afm\_group.AFMGroup*) – The path to the data file or an instance of *AFMGroup*
- **meta\_override** (*dict*) – Dictionary with metadata that is used when loading the data in *path*.
- **callback** (*callable* or *None*) – A method that accepts a float between 0 and 1 to externally track the process of loading the data.
- **data\_classes\_by\_modality** (*dict*) – Override the default *AFMData* class to use for managing the data (see *default\_data\_classes\_by\_modality*): This is e.g. used by *nanite:index* to pass *Indentation* (which is a subclass of the default *AFMForceDistance*) for handling “force-indentation” data.

## Inheritance

AFMQMap

```
static feat_core_data_height_base_point_um(afmdata)
```

Compute the lowest height (measured)

```
static feat_core_data_piezo_range_um(afmdata)
```

Compute peak-to-peak piezo range

```
static feat_core_data_scan_order(afmdata)
```

Return the enumeration of the dataset

```
get_coords(which='px')
```

Get the qmap coordinates for each curve in *AFMQMap.group*

**Parameters** **which** (*str*) – “px” for pixels or “um” for microns.

```
get_qmap(feature, qmap_only=False)
```

Return the quantitative map for a feature

### Parameters

- **feature** (*str*) – Feature to compute map for (see *QMap.features*)
- **qmap\_only** – Only return the quantitative map data, not the coordinates

### Returns



- **x, y** (*1d ndarray*) – Only returned if *qmap\_only* is False; Pixel grid coordinates along x and y
- **qmap** (*2d ndarray*) – Quantitative map

**property extent**extent (x1, x2, y1, y2) [ $\mu\text{m}$ ]**features**

Available features

**group**AFM data (instance of *afmformats.afm\_group.AFMGroup*)**property shape**

shape of the map [px]

**Variables**

- *unit\_scales*

**afmformats.afm\_qmap.unit\_scales**

dict() -&gt; new empty dictionary dict(mapping) -&gt; new dictionary initialized from a mapping object's

(key, value) pairs

**dict(iterable)** -> new dictionary initialized as if via:  $d = \{ \}$  for k, v in iterable: $d[k] = v$ **dict(\*\*kwargs)** -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)

{'': 1, 'k': 1000.0, 'm': 0.001, 'n': 1e-09, 'p': 1e-12, 'μ': 1e-06}

**3.1.6 afmformats.errors**

- *Classes*

**Classes**

- *AFMFileFormatError*: Common base class for all exceptions
- *DataFileBrokenError*: Common base class for all exceptions
- *FileFormatMetadataError*: Common base class for all exceptions
- *FileFormatNotSupportedError*: Common base class for all exceptions
- *InvalidFileFormatError*: Common base class for all exceptions
- *MissingMetadataError*: Common base class for all exceptions

**class afmformats.errors.AFMFileFormatError**

## Inheritance

AFMFileFormatError

```
class afmformats.errors.DataFileBrokenError
```

## Inheritance



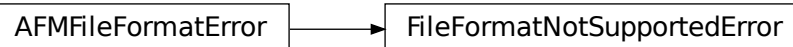
```
class afmformats.errors.FileFormatMetadataError
```

## Inheritance



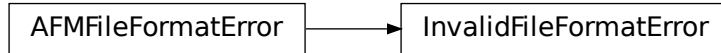
```
class afmformats.errors.FileFormatNotSupportedError
```

## Inheritance



```
class afmformats.errors.InvalidFileFormatError
```

## Inheritance

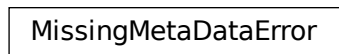


**class** `afmformats.errors.MissingMetadataError`(*meta\_keys*, \*args, \*\*kwargs)

Special error class for missing metadata

The missing metadata keys are stored in the `meta_keys` property.

## Inheritance



### **meta\_keys**

List of missing metadata keys

### 3.1.7 `afmformats.fmt_hdf5`

- *Functions*
- *Classes*

## Functions

- `load_hdf5()`: Loads HDF5 files as exported by afmformats

`afmformats.fmt_hdf5.load_hdf5`(*path\_or\_h5*, *callback=None*, *meta\_override=None*)

Loads HDF5 files as exported by afmformats

The HDF5 format is self explanatory. The root attributes contain the version of afmformats used to create it. For each curve, one group is created, named according to “0”, “1”, ... “9”, “10”, “11”, etc. The attributes of each group are key-value pairs defined in `afmformats.meta.KEYS_VALID`. The group contains datasets named according to `afmformats.afm_data.known_columns` and have the attribute “unit” with the corresponding value in `afmformats.afm_data.column_units`.

### **Parameters**

- **path\_or\_h5** (*str* or *pathlib.Path* or *h5py.Group*) – path to HDF5 file or an HDF5 group

- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.
- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see [\*afmformats.meta.META\\_FIELDS\*](#))

## Notes

In case *path\_or\_h5* is a `h5py.Group` object, the “path” metadata variable will always be set to the path of the original HDF5 file. Keep this in mind if you think about storing multiple datasets (each containing multiple curves) in one HDF5 file (bad idea).

## Classes

- *H5DictReader*: Undocumented.

**class** `afmformats.fmt_hdf5.H5DictReader`(*path\_or\_h5*, *enum\_key*)

Read-only HDF5-based dictionary for arrays

### Parameters

- **path\_or\_h5** (*str* or *pathlib.Path* or *h5py.Group*) – Path to HDF5 file or an HDF5 group
- **enum\_key** (*str*) – Name of the subgroup in *path\_or\_h5* that contains the data of the dictionary

## Inheritance

H5DictReader

### 3.1.8 `afmformats.fmt_igor`

- *Functions*

## Functions

- `load_igor()`: Load Asylum Research (Igor) binarywave .ibw files

`afmformats.fmt_igor.load_igor(path, callback=None, meta_override=None)`

Load Asylum Research (Igor) binarywave .ibw files

The raw data are loaded with the Python module “igor” (<http://blog.tremily.us/posts/igor/>). The way column labels are assigned to the data is kind of hacky. The metadata assignment is largely guessed.

Test data were provided by Nicolas Hauck [HSC+18].

### Parameters

- **path** (*str* or *pathlib.Path*) – path to in .ibw data file
- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.
- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see `afmformats.meta.META_FIELDS`)

## 3.1.9 afmformats.fmt\_jpk

- *Submodules*
- *Functions*

## Submodules

`afmformats.fmt_jpk.jpk_data`

- *Functions*
- *Classes*
- *Variables*

## Functions

- `find_column_dat()`: Find a column in a list of strings
- `load_dat_raw()`: Load data from binary JPK .dat files
- `load_dat_unit()`: Load data from a JPK .dat file with a specific calibration slot

`afmformats.fmt_jpk.jpk_data.find_column_dat(loc_list, column)`

Find a column in a list of strings

### Parameters

- **loc\_list** (*list of str*) – The segment’s data location list (within the archive), e.g.  
`['segments/0/channels/height.dat', 'segments/0/channels/vDeflection.dat', 'segments/0/channels/strainGaugeHeight.dat']`

- **column** (*str*) – afmformats column name `afmformats.afm_data.known_columns`

#### Returns

- **name** (*str*) – Matched column name from `JPK_COLUMNS`, e.g. “strainGaugeHeight”
- **slot** (*str*) – Default slot location from `JPK_SLOTS`
- **loc** (*str*) – Matched data location in zip file, e.g. “segments/0/channels/strainGaugeHeight.dat”

`afmformats.fmt_jpk.jpk_data.load_dat_raw(fd, name, properties)`

Load data from binary JPK .dat files

#### Parameters

- **fd** (*file*) – Open .dat file
- **name** (*str*) – Name of the data to read (required for scale conversions) (valid options are values in `JPK_COLUMNS`)
- **properties** (*dict*) – Property dictionary metadata (see also `JPKReader._get_index_segment_properties()`)

**Returns data** – A numpy array with the raw data.

**Return type** 1d ndarray

#### Notes

This method tries to correctly determine the data type of the binary data and scales it with the `data.encoder.scaling` values given in the header files.

**See also:**

`load_dat_unit` Includes conversion to useful units

`afmformats.fmt_jpk.jpk_data.load_dat_unit(fd, name, properties, slot='default')`

Load data from a JPK .dat file with a specific calibration slot

#### Parameters

- **fd** (*file*) – Open .dat file
- **name** (*str*) – Name of the data to read (required for scale conversions) (valid options are values in `JPK_COLUMNS`)
- **properties** (*dict*) – Property dictionary metadata (see also `JPKReader._get_index_segment_properties()`)
- **slot** (*str*) – The .dat files in the JPK measurement zip files come with different calibration slots. Valid values are
  - For the height of the piezo crystal during measurement (the piezo height is not as accurate as the measured height from the height sensor; the piezo movement is not linear): “height.dat”: “volts”, “nominal”, “calibrated”
  - For the measured height of the cantilever: “strainGaugeHeight.dat”: “volts”, “nominal”, “absolute” “measuredHeight.dat”: “volts”, “nominal”, “absolute” “capacitiveSensorHeight”: “volts”, “nominal”, “absolute” (they are all the same)
  - For the recorded cantilever deflection: “vDeflection.dat”: “volts”, “distance”, “force”

#### Returns

- **data** (*1d ndarray*) – A numpy array containing the scaled data.
- **unit** (*str*) – A string representing the metric unit of the data.
- **name** (*str*) – The name of the data column.

## Notes

The raw data (see `load_dat_raw`) is usually stored in “volts” and needs to be converted to e.g. “force” for “vDeflection” or “nominal” for “strainGaugeHeight”. The conversion parameters (offset, multiplier) are stored in the header files and they are not stored separately for each slot, but the conversion parameters are stored relative to the slots. For instance, to compute the “force” slot from the raw “volts” data, one first needs to compute the “distance” slot. This conversion is taken care of by this method.

This is an example header:

```
channel.vDeflection.data.file.name=channels/vDeflection.dat          chan-
nel.vDeflection.data.file.format=raw          channel.vDeflection.data.type=short      chan-
nel.vDeflection.data.encoder.type=signedshort channel.vDeflection.data.encoder.scaling.type=linear
channel.vDeflection.data.encoder.scaling.style=offsetmultiplier channel.vDeflection.data.encoder.scaling.offset=-
0.00728873489143207 channel.vDeflection.data.encoder.scaling.multiplier=3.0921021713588157E-
4          channel.vDeflection.data.encoder.scaling.unit.type=metric-unit          chan-
nel.vDeflection.data.encoder.scaling.unit.unit=V          channel.vDeflection.channel.name=vDeflection
channel.vDeflection.conversion-set.conversions.list=distance force channel.vDeflection.conversion-
set.conversions.default=force          channel.vDeflection.conversion-set.conversions.base=volts
channel.vDeflection.conversion-set.conversion.volts.name=Volts      channel.vDeflection.conversion-
set.conversion.volts.defined=false          channel.vDeflection.conversion-
set.conversion.distance.name=Distance          channel.vDeflection.conversion-
set.conversion.distance.defined=true          channel.vDeflection.conversion-
set.conversion.distance.type=simple          channel.vDeflection.conversion-
set.conversion.distance.comment=Distance          channel.vDeflection.conversion-
set.conversion.distance.base-calibration-slot=volts          channel.vDeflection.conversion-
set.conversion.distance.calibration-slot=distance          channel.vDeflection.conversion-
set.conversion.distance.scaling.type=linear          channel.vDeflection.conversion-
set.conversion.distance.scaling.style=offsetmultiplier          channel.vDeflection.conversion-
set.conversion.distance.scaling.offset=0.0          channel.vDeflection.conversion-
set.conversion.distance.scaling.multiplier=7.000143623002982E-8 channel.vDeflection.conversion-
set.conversion.distance.scaling.unit.type=metric-unit          channel.vDeflection.conversion-
set.conversion.distance.scaling.unit.unit=m          channel.vDeflection.conversion-
set.conversion.force.name=Force channel.vDeflection.conversion-set.conversion.force.defined=true
channel.vDeflection.conversion-set.conversion.force.type=simple channel.vDeflection.conversion-
set.conversion.force.comment=Force          channel.vDeflection.conversion-set.conversion.force.base-
calibration-slot=distance          channel.vDeflection.conversion-set.conversion.force.calibration-
slot=force          channel.vDeflection.conversion-set.conversion.force.scaling.type=linear
channel.vDeflection.conversion-set.conversion.force.scaling.style=offsetmultiplier
channel.vDeflection.conversion-set.conversion.force.scaling.offset=0.0
channel.vDeflection.conversion-set.conversion.force.scaling.multiplier=0.043493666407368466
channel.vDeflection.conversion-set.conversion.force.scaling.unit.type=metric-unit
channel.vDeflection.conversion-set.conversion.force.scaling.unit.unit=N
```

To convert from the raw “volts” data to force data, these steps are performed:

- Convert from “volts” to “distance” first, because the “base-calibration-slot” for force is “distance”.  

$$\text{distance} = \text{volts} * 7.000143623002982E-8 + 0.0$$
- Convert from “distance” to “force”:

```
force = distance*0.043493666407368466 + 0.0
```

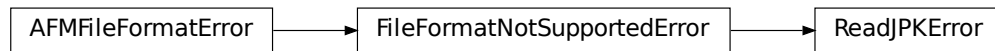
The multipliers shown above are the values for sensitivity and spring constant: sensitivity = 7.000143623002982E-8 m/V spring\_constant = 0.043493666407368466 N/m

## Classes

- [\*ReadJPKError\*](#): Common base class for all exceptions

```
class afmformats.fmt_jpk.jpk_data.ReadJPKError
```

## Inheritance



## Variables

- [\*JPk\\_COLUMNS\*](#)
- [\*JPk\\_SLOTS\*](#)
- [\*JPk\\_UNITS\*](#)

```
afmformats.fmt_jpk.jpk_data.JPK_COLUMNS
```

Maps afmformats column names to JPK column names

```
{'force': ['vDeflection'],
 'height (measured)': ['strainGaugeHeight',
                       'capacitiveSensorHeight',
                       'measuredHeight'],
 'height (piezo)': ['height', 'head-height']}
```

```
afmformats.fmt_jpk.jpk_data.JPK_SLOTS
```

Maps afmformats column names to default JPK normalization slots

```
{'force': 'force',
 'height (measured)': 'nominal',
 'height (piezo)': 'calibrated'}
```

```
afmformats.fmt_jpk.jpk_data.JPK_UNITS
```

Maps afmformats column names to default JPK units

```
{'force': 'N', 'height (measured)': 'm', 'height (piezo)': 'm'}
```



`afmformats.fmt_jpk.jpk_meta`

- *Classes*
- *Variables*

## Classes

- *ReadJPKMetaKeyError*: Common base class for all exceptions

`class afmformats.fmt_jpk.jpk_meta.ReadJPKMetaKeyError`

## Inheritance



## Variables

- *get\_primary\_meta\_recipe*
- *get\_secondary\_meta\_recipe*

`afmformats.fmt_jpk.jpk_meta.get_primary_meta_recipe`

```
<functools._lru_cache_wrapper object at 0x7fe63de6f438>
```

`afmformats.fmt_jpk.jpk_meta.get_secondary_meta_recipe`

```
<functools._lru_cache_wrapper object at 0x7fe63de6f390>
```

`afmformats.fmt_jpk.jpk_reader`

- *Classes*

## Classes

- *ArchiveCache*: Archive cache for fast access to zip data
- *JPKReader*: Undocumented.

**class** `afmformats.fmt_jpk.jpk_reader.ArchiveCache`

Archive cache for fast access to zip data

If every *JPKReader* has its own instance of *ZipFile*, then on macOS (and possibly other OSes), we might run into an `OSError; [Errno 24] Too many open files` (<https://github.com/AFM-analysis/afmformats/issues/10>). The problem is, that if we don't leave the *ZipFile*, we have to re-open it every time we want to access some data. This is a huge overhead.

The solution is *ArchiveCache*, which keeps a reference to the last `max_archives=32` archives and closes the ones that were used least.

## Inheritance

ArchiveCache

**static** `get(zip_path)`

Return the (possibly cached) *ZipFile* object for *zip\_path*

**class** `afmformats.fmt_jpk.jpk_reader.JPKReader(path)`

## Inheritance

JPKReader

**get\_data**(*column*, *index*, *segment=None*)

Return data for a given column, index, or segment

### Parameters

- **column** (*str*) – Valid column from `afmformats.afm_data.known_columns`
- **index** (*int*) – Curve index in the current archive
- **segment** (*int* or *None*) – Segment index for chosen curve index

**Returns** `data` – Column data

**Return type** 1d ndarray

**get\_index\_numbers()**

Return int array with available index numbers

The numbers is what we refer to as “enum” in afmformats. Sometimes individual curves are missing from JPK files. These have to be correctly indexed.

**get\_index\_path(*index*)**

Return the path in the zip file for a specific curve index

**get\_index\_segment\_numbers(*index*)**

Return available segment numbers for an index

**get\_index\_segment\_path(*index*, *segment*)**

Return the path in the zip file for a specific index and segment

**get\_metadata(*index*, *segment*=None)**

Return the metadata for a specific index and segment

#### Parameters

- **index** (*int*) – Curve index; For “single” hierarchy files, this should be 0.
- **segment** (*int or None*) – If None, then all segment-specific properties (e.g. approach and retract) are returned.

**property files**

List of files and folders in the archive

**property hierarchy**

Format hierarchy (“single” or “indexed”)

## Functions

- **load\_jpk()**: Loads JPK Instruments data files

`afmformats.fmt_jpk.load_jpk(path, callback=None, meta_override=None)`

Loads JPK Instruments data files

These files are zip files containing java property files and integer-encoded binary data. The property files include recipes on how to convert the raw integer data to SI units.

#### Parameters

- **path** (*str or pathlib.Path*) – path to JPK data file
- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.
- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see `afmformats.meta.META_FIELDS`)

### 3.1.10 afmformats.fmt\_ntmdt\_txt

- *Functions*

#### Functions

- `load_txt()`: Load text files exported by the NT-MDT Nova software

`afmformats.fmt_ntmdt_txt.load_txt(path, callback=None, meta_override=None)`

Load text files exported by the NT-MDT Nova software

The columns are assumed to be: height (piezo) [nm], Deflection during approach (nA), Deflection during retraction (nA). The sensitivity in `meta_override` should be given in [m/A] (even though it is displayed as [m/V]).

This loader removes constant-value padding at the beginning of the data columns, an artifact that is sometimes introduced during data export. There are no metadata in this file format.

Test data were provided by Yuri Efremov [Efr20] [EBKS15].

Note that support for the original .mdt files is currently (2020) not possible. There exist binary readers for ntmdt files ([https://github.com/kaitai-io/kaitai\\_struct/](https://github.com/kaitai-io/kaitai_struct/)), but this does not work for [exemplary data](#). If the NT-MDT Nova software is not available, it should still be possible to load the data with [Ggyddion](#) and export it to something afmformats understands.

#### Parameters

- **path** (*str* or *pathlib.Path* or *io.TextIOBase*) – path to an ntmdt-exported .txt file
- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.
- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see `afmformats.meta.META_FIELDS`)

### 3.1.11 afmformats.fmt\_tab

- *Functions*

#### Functions

- `load_tab()`: Loads tab-separated-value files as exported by afmformats

`afmformats.fmt_tab.load_tab(path, callback=None, meta_override=None)`

Loads tab-separated-value files as exported by afmformats

This is a simple tab-separated values files. The metadata may be present at the beginning of the file, commented out, as a json dump in a “BEGIN METADATA” - “END METADATA” block. The column data is listed below as a simple table.

#### Parameters

- **path** (*str* or *pathlib.Path* or *io.TextIOBase*) – path to a .tab file
- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.

- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see `afmformats.meta.META_FIELDS`)

### 3.1.12 afmformats.fmt\_workshop

- *Submodules*

#### Submodules

`afmformats.fmt_workshop.ws_map`

- *Functions*

#### Functions

- `load_map()`: Load a set of zipped csv AFM workshop data

`afmformats.fmt_workshop.ws_map.load_map(path, callback=None, meta_override=None)`

Load a set of zipped csv AFM workshop data

If you are recording quantitative force-maps (i.e. multiple curves on an x-y-grid) with AFM workshop setups, then you might have realized that you get *multiple* .csv files (one file per indentation) instead of *one* file that contains all the data (as you might be accustomed to from other manufacturers). Since afmformats expects one file per measurement, it would not be straight forward to obtain a properly enumerated quantitative imaging group.

This function offers a workaround - it loads a zip archive created from the the .csv files.

The files are structured like this:

```
Force-Distance Curve
File Format:      3

Date:    Wednesday, August 1, 2018
Time:    1:07:47 PM
Mode:    Mapping
Point:    16
X, um:    27.250000
Y, um:    27.250000

Extend Z-Sense(nm),Extend T-B(V),Retract Z-Sense(nm),Retract T-B(V)
13777.9288,0.6875,14167.9288,1.0917
13778.9288,0.6874,14166.9288,1.0722
13779.9288,0.6876,14165.9288,1.0693
13780.9288,0.6877,14164.9288,1.0824
13781.9288,0.6875,14163.9288,1.0989
...
```

Please make sure that the `Point` is enumerated from 1 onwards (and matches the alphanumerical order of the files in the archive) and that `Mode` is `Mapping`. The `X` and `Y` coordinates can be used by e.g. `PyJibe` to display `QMap` data on a grid.

#### Parameters

- **path** (*str* or *pathlib.Path*) – path to zip file containing AFM workshop .csv files
- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.
- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see *afmformats.meta.META\_FIELDS*)

`afmformats.fmt_workshop.ws_single`

- *Functions*

## Functions

- *load\_csv()*: Load csv data from AFM workshop

`afmformats.fmt_workshop.ws_single.load_csv(path, callback=None, meta_override=None, mode='single')`  
Load csv data from AFM workshop

The files are structured like this:

```
Force-Distance Curve
File Format:      3

Date:    Wednesday, August 1, 2018
Time:    1:07:47 PM
Mode:    Single
Point:   1
X, um:   27.250000
Y, um:   27.250000

Extend Z-Sense(nm),Extend T-B(V),Retract Z-Sense(nm),Retract T-B(V)
13777.9288,0.6875,14167.9288,1.0917
13778.9288,0.6874,14166.9288,1.0722
13779.9288,0.6876,14165.9288,1.0693
13780.9288,0.6877,14164.9288,1.0824
13781.9288,0.6875,14163.9288,1.0989
...
```

The data for testing was kindly provided by Peter Eaton (afmhelp.com).

#### Parameters

- **path** (*str* or *pathlib.Path* or *io.TextIOBase*) – data file or an open file in text (not bytes) mode
- **callback** (*callable*) – function for progress tracking; must accept a float in [0, 1] as an argument.

- **meta\_override** (*dict*) – if specified, contains key-value pairs of metadata that are used when loading the files (see [afmformats.meta.META\\_FIELDS](#))
- **mode** (*str*) – curve mode to expect (either “single” or “mapping”); if an unexpected mode is found, AFMWorkshopFormatWarning is issued

### 3.1.13 afmformats.formats

- *Functions*
- *Classes*
- *Variables*

#### Functions

- *find\_data()*: Recursively find valid AFM data files
- *get\_recipe()*: Return the file format recipe for a given path
- *load\_data()*: Load AFM data

`afmformats.formats.find_data(path, modality=None)`

Recursively find valid AFM data files

##### Parameters

- **path** (*str* or *pathlib.Path*) – file or directory
- **modality** (*str*) – modality of the measurement (“force-distance”)

**Returns** *file\_list* – list of valid AFM data files

**Return type** list of *pathlib.Path*

`afmformats.formats.get_recipe(path, modality=None)`

Return the file format recipe for a given path

##### Parameters

- **path** (*str* or *pathlib.Path*) – file or directory
- **modality** (*str*) – modality of the measurement (“force-distance”)

**Returns** *recipe* – file format recipe

**Return type** *AFMFormatRecipe*

`afmformats.formats.load_data(path, meta_override=None, modality=None, data_classes_by_modality=None, diskcache=False, callback=None)`

Load AFM data

##### Parameters

- **path** (*str* or *pathlib.Path*) – Path to AFM data file
- **meta\_override** (*dict*) – Metadata dictionary that overrides experimental metadata
- **modality** (*str*) – Which acquisition modality to use (currently only “force-distance”)

- **data\_classes\_by\_modality** (*dict*) – Override the default AFMData class to use for managing the data (see *default\_data\_classes\_by\_modality*): This is e.g. used by nanite:index to pass *Indentation* (which is a subclass of the default *AFMForceDistance*) for handling “force-indentation” data.
- **diskcache** (*bool*) – Whether to use caching (not implemented)
- **callback** (*callable*) – A method that accepts a float between 0 and 1 to externally track the process of loading the data

**Returns** *afm\_list* – List where each element is on AFMData curve

**Return type** list of afmformats.afm\_data.AFMData

## Classes

- *AFMFormatRecipe*: Undocumented.

**class** afmformats.formats.AFMFormatRecipe(*recipe*)

A wrapper class for file format recipes

**Parameters** *recipe* (*dict*) – file format recipe

## Inheritance

AFMFormatRecipe

**detect**(*path*)

Determine whether *path* can be opened with this recipe

**Returns** *valid* – True if *path* is openable, False otherwise.

**Return type** *bool*

## Notes

If the underlying recipe does not implement a “detect” function, then only the file extension is checked.

**property** *descr*

description of file format

**property** *loader*

method for loading the data

**property** *maker*

who introduced the file format

**property** *modality*

key describing the AFM recording modality



**property suffix**  
file format suffix

## Variables

- *default\_data\_classes\_by\_modality*
- *formats\_available*
- *formats\_by\_suffix*
- *formats\_by\_modality*
- *supported\_extensions*

**afmformats.formats.default\_data\_classes\_by\_modality**  
dictionary with default data classes for each modality

```
{'force-distance': <class 'afmformats.mod_force_distance.AFMForceDistance'>}
```

**afmformats.formats.formats\_available**  
available/supported file formats

```
[<AFMFormatRecipe from 'afmformats.fmt_hdf5' at 0x7fe63e587be0>,
<AFMFormatRecipe from 'afmformats.fmt_igor' at 0x7fe63e587c18>,
<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de840f0>,
<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de84128>,
<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de84160>,
<AFMFormatRecipe from 'afmformats.fmt_ntmdt_txt' at 0x7fe63de84198>,
<AFMFormatRecipe from 'afmformats.fmt_tab' at 0x7fe63de841d0>,
<AFMFormatRecipe from 'afmformats.fmt_workshop.ws_map' at 0x7fe63de84208>,
<AFMFormatRecipe from 'afmformats.fmt_workshop.ws_single' at 0x7fe63de84240>]
```

**afmformats.formats.formats\_by\_suffix**  
available file formats in a dictionary with suffix keys

```
{'.csv': [<AFMFormatRecipe from 'afmformats.fmt_workshop.ws_single' at 0x7fe63de84240>],
'.h5': [<AFMFormatRecipe from 'afmformats.fmt_hdf5' at 0x7fe63e587be0>],
'.ibw': [<AFMFormatRecipe from 'afmformats.fmt_igor' at 0x7fe63e587c18>],
'.jpk-force': [<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de840f0>],
'.jpk-force-map': [<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de84128>],
'.jpk-qi-data': [<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de84160>],
'.tab': [<AFMFormatRecipe from 'afmformats.fmt_tab' at 0x7fe63de841d0>],
'.txt': [<AFMFormatRecipe from 'afmformats.fmt_ntmdt_txt' at 0x7fe63de84198>],
'.zip': [<AFMFormatRecipe from 'afmformats.fmt_workshop.ws_map' at 0x7fe63de84208>]}
```

**afmformats.formats.formats\_by\_modality**  
available file formats in a dictionary for each modality

```
{'force-distance': [<AFMFormatRecipe from 'afmformats.fmt_hdf5' at 0x7fe63e587be0>,
<AFMFormatRecipe from 'afmformats.fmt_igor' at 0x7fe63e587c18>,
<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de840f0>,
<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de84128>,
<AFMFormatRecipe from 'afmformats.fmt_jpk' at 0x7fe63de84160>,
```

(continues on next page)

(continued from previous page)

```
<AFMFormatRecipe from 'afmformats.fmt_ntmdt_txt' at 0x7fe63de84198>,
<AFMFormatRecipe from 'afmformats.fmt_tab' at 0x7fe63de841d0>,
<AFMFormatRecipe from 'afmformats.fmt_workshop.ws_map' at 0x7fe63de84208>,
<AFMFormatRecipe from 'afmformats.fmt_workshop.ws_single' at 0x7fe63de84240>]]}
```

**afmformats.formats.supported\_extensions**

list of supported extensions

```
['.csv',
 '.h5',
 '.ibw',
 '.jpk-force',
 '.jpk-force-map',
 '.jpk-qi-data',
 '.tab',
 '.txt',
 '.zip']
```

### 3.1.14 afmformats.lazy\_loader

- *Classes*

#### Classes

- *LazyData*: Lazily load data from function and kwargs

**class** afmformats.lazy\_loader.LazyData

Lazily load data from function and kwargs

The idea is that the experimental data does not have to be loaded before the user requests it. Furthermore, this reduces the memory footprint (not all data are loaded).

#### Inheritance

LazyData

**set\_lazy\_loader**(*column, func, kwargs*)

Add a lazy loader

#### Parameters

- **column** (*str*) – Column for which to register the loader
- **func** (*callable*) – Function to call to get the data
- **kwargs** – Keyword arguments to **func**

### 3.1.15 afmformats.meta

- *Functions*
- *Classes*
- *Variables*

#### Functions

- **parse\_time()**: Convert a time string to “HH:MM:SS.S”

**afmformats.meta.parse\_time**(*value*)

Convert a time string to “HH:MM:SS.S”

- leading zeros are added where necessary
- trailing zeros after “.” are stripped
- trailing “.” is stripped

e.g.

- “6:15:22 PM” -> “18:15:22”
- “6:15:22.00 AM” -> “06:15:22”
- “6:02:22.010 AM” -> “06:02:22.01”

#### Classes

- **MetaDataMissingError**: Raised when meta data is missing
- **MetaData**: Management of meta data variables

**class afmformats.meta.MetaDataMissingError**

Raised when meta data is missing

#### Inheritance

MetaDataMissingError

**class** `afmformats.meta.MetaData(*args, **kwargs)`

Management of meta data variables

Valid key names are defined in `afmformats.meta.KEYS_VALID`.

## Inheritance

MetaData

**as\_dict()**

Convert to real dictionary

This is needed e.g. for `self.items` such that `json.dump` works in combination with `LazyMetaValue` (which is not JSON serializable)

**copy()**

Create a copy of the metadata

**Returns** `mdc` – Copy of the MetaData class (LazyMetaValue not copied)

**Return type** `MetaData`

**get**(`k`, `d`) → `D[k]` if `k` in `D`, else `d`. `d` defaults to `None`.

**get\_summary()**

Convenience function returning the meta data summary

Returns a dict of dicts with keys matching the `DEF_*` dicts. Unset values are returned as `np.nan`.

**items()** → a set-like object providing a view on `D`'s items

**update**(`[E]`, `**F`) → `None`. Update `D` from dict/iterable `E` and `F`.

If `E` is present and has a `.keys()` method, then does: for `k` in `E`: `D[k] = E[k]` If `E` is present and lacks a `.keys()` method, then does: for `k, v` in `E`: `D[k] = v` In either case, this is followed by: for `k` in `F`: `D[k] = F[k]`

**values()** → an object providing a view on `D`'s values

## Variables

- `META_FIELDS`
- `DEF_ALL`
- `KEYS_VALID`

`afmformats.meta.META_FIELDS`

Compendium of all allowed meta data keys, sorted by topic, and including units and validation methods

```
{'acquisition': {'feedback mode': ['Feedback mode',  
                                   ''],  
                 <function vd_str_in.<locals>.str_in at 0x7fe64206c598>],
```

(continues on next page)

(continued from previous page)

```

        'imaging mode': ['Imaging modality',
            '',
            <function vd_str_in.<locals>.str_in at 0x7fe64206c7b8>],
        'rate approach': ['Sampling rate (approach)',
            'Hz',
            <class 'float'>],
        'rate retract': ['Sampling rate (retract)',
            'Hz',
            <class 'float'>],
        'sensitivity': ['Sensitivity', 'm/V', <class 'float'>],
        'setpoint': ['Active feedback loop setpoint',
            'N',
            <class 'float'>],
        'spring constant': ['Cantilever spring constant',
            'N/m',
            <class 'float'>]],
    'dataset': {'duration': ['Duration', 's', <class 'float'>],
        'enum': ['Dataset index within the experiment',
            '',
            <function fint at 0x7fe64206c2f0>],
        'point count': ['Size of the dataset in points',
            '',
            <function fint at 0x7fe64206c2f0>],
        'speed approach': ['Piezo speed (approach)',
            'm/s',
            <class 'float'>],
        'speed retract': ['Piezo speed (retract)', 'm/s', <class 'float'>],
        'z range': ['Axial piezo range', 'm', <class 'float'>]],
    'qmap': {'grid center x': ['Horizontal center of grid', 'm', <class 'float'>],
        'grid center y': ['Vertical center of grid', 'm', <class 'float'>],
        'grid index x': ['Horizontal grid position index',
            '',
            <function fint at 0x7fe64206c2f0>],
        'grid index y': ['Vertical grid position index',
            '',
            <function fint at 0x7fe64206c2f0>],
        'grid shape x': ['Horizontal grid shape',
            'px',
            <function fint at 0x7fe64206c2f0>],
        'grid shape y': ['Vertical grid shape',
            'px',
            <function fint at 0x7fe64206c2f0>],
        'grid size x': ['Horizontal grid image size', 'm', <class 'float'>],
        'grid size y': ['Vertical grid image size', 'm', <class 'float'>],
        'position x': ['Horizontal position', 'm', <class 'float'>],
        'position y': ['Vertical position', 'm', <class 'float'>]],
    'setup': {'instrument': ['Instrument', '', <class 'str'>],
        'software': ['Acquisition software', '', <class 'str'>],
        'software version': ['Acquisition software version',
            '',
            <class 'str'>]],

```

(continues on next page)

(continued from previous page)

```
'storage': {'curve id': ['Curve identifier', '', <class 'str'>],
            'date': ['Recording date', '', <class 'str'>],
            'format': ['File format', '', <class 'str'>],
            'path': ['Path', '', <class 'pathlib.Path'>],
            'session id': ['Dataset identifier', '', <class 'str'>],
            'time': ['Recording time', '', <class 'str'>]}
```

**afmformats.meta.DEF\_ALL**

A dictionary for all metadata definitions

```
{'curve id': ['Curve identifier', '', <class 'str'>],
 'date': ['Recording date', '', <class 'str'>],
 'duration': ['Duration', 's', <class 'float'>],
 'enum': ['Dataset index within the experiment',
          '',
          <function fint at 0x7fe64206c2f0>],
 'feedback mode': ['Feedback mode',
                   '',
                   <function vd_str_in.<locals>.str_in at 0x7fe64206c598>],
 'format': ['File format', '', <class 'str'>],
 'grid center x': ['Horizontal center of grid', 'm', <class 'float'>],
 'grid center y': ['Vertical center of grid', 'm', <class 'float'>],
 'grid index x': ['Horizontal grid position index',
                  '',
                  <function fint at 0x7fe64206c2f0>],
 'grid index y': ['Vertical grid position index',
                  '',
                  <function fint at 0x7fe64206c2f0>],
 'grid shape x': ['Horizontal grid shape',
                  'px',
                  <function fint at 0x7fe64206c2f0>],
 'grid shape y': ['Vertical grid shape',
                  'px',
                  <function fint at 0x7fe64206c2f0>],
 'grid size x': ['Horizontal grid image size', 'm', <class 'float'>],
 'grid size y': ['Vertical grid image size', 'm', <class 'float'>],
 'imaging mode': ['Imaging modality',
                  '',
                  <function vd_str_in.<locals>.str_in at 0x7fe64206c7b8>],
 'instrument': ['Instrument', '', <class 'str'>],
 'path': ['Path', '', <class 'pathlib.Path'>],
 'point count': ['Size of the dataset in points',
                 '',
                 <function fint at 0x7fe64206c2f0>],
 'position x': ['Horizontal position', 'm', <class 'float'>],
 'position y': ['Vertical position', 'm', <class 'float'>],
 'rate approach': ['Sampling rate (approach)', 'Hz', <class 'float'>],
 'rate retract': ['Sampling rate (retract)', 'Hz', <class 'float'>],
 'sensitivity': ['Sensitivity', 'm/V', <class 'float'>],
 'session id': ['Dataset identifier', '', <class 'str'>],
 'setpoint': ['Active feedback loop setpoint', 'N', <class 'float'>],
 'software': ['Acquisition software', '', <class 'str'>],
```

(continues on next page)

(continued from previous page)

```
'software version': ['Acquisition software version', '', <class 'str'>],
'speed approach': ['Piezo speed (approach)', 'm/s', <class 'float'>],
'speed retract': ['Piezo speed (retract)', 'm/s', <class 'float'>],
'spring constant': ['Cantilever spring constant', 'N/m', <class 'float'>],
'time': ['Recording time', '', <class 'str'>],
'z range': ['Axial piezo range', 'm', <class 'float'>]}
```

**afmformats.meta.KEYS\_VALID**

List of all valid meta data keys

```
['curve id',
'date',
'duration',
'enum',
'feedback mode',
'format',
'grid center x',
'grid center y',
'grid index x',
'grid index y',
'grid shape x',
'grid shape y',
'grid size x',
'grid size y',
'imaging mode',
'instrument',
'path',
'point count',
'position x',
'position y',
'rate approach',
'rate retract',
'sensitivity',
'session id',
'setpoint',
'software',
'software version',
'speed approach',
'speed retract',
'spring constant',
'time',
'z range']
```

### 3.1.16 afmformats.mod\_force\_distance

- *Classes*

#### Classes

- *AFMForceDistance*: Base class for AFM force-distance data
- *Segment*: Simple wrapper around dict-like *data* to expose a single segment

**class** afmformats.mod\_force\_distance.**AFMForceDistance**(*data*, *metadata*, *diskcache=False*)

Base class for AFM force-distance data

A force-distance dataset consists of an approach and a retract curve.

Initialization

#### Parameters

- **data** (*dict-like*) – Experimental data
- **metadata** (*dict*) – Metadata
- **diskcache** (*bool*) – TODO

#### Inheritance



#### **property** **appr**

Dictionary-like interface to the approach segment

#### **property** **modality**

Imaging modality

#### **property** **retr**

Dictionary-like interface to the retract segment

**class** afmformats.mod\_force\_distance.**Segment**(*raw\_data*, *data*, *which='approach'*)

Simple wrapper around dict-like *data* to expose a single segment



## Inheritance

Segment

### which

The segment type (approach or retract)

### 3.1.17 afmformats.parse\_funcs

- *Functions*

#### Functions

- *fbool()*: boolean
- *fint()*: integer
- *vd\_str\_in()*: Return a validator that tests whether a string is in a list

`afmformats.parse_funcs.fbool(value)`  
boolean

`afmformats.parse_funcs.fint(value)`  
integer

`afmformats.parse_funcs.vd_str_in(alist)`  
Return a validator that tests whether a string is in a list



## CHANGELOG

List of changes in-between afmformats releases.

### 4.1 version 0.14.3

- enh: implement LazyMetaValue to speed-up loading JPK files
- enh: perform caching in LazyData (minor speed-up)

### 4.2 version 0.14.2

- fix: partially revert breaking change in 0.14.0 (“imaging mode” was previously used when exporting data in PyJibe and this metadata key should not change)
- fix: add detect function for HDF5 format (afmformats attempted to load nanite rating containers)
- enh: added height span for QMap data
- tests: add .tab and .h5 test files
- ref: renamed QMap feature “lowest height” to “height base point”
- ref: changed prefix to “data” for all QMap data

### 4.3 version 0.14.1

- fix: populate AFMQMap grid metadata for AFM workshop format (#12)
- enh: speed-up QMap computation by decorator-based caching
- ref: move QMap grid index computation to MetaData class

## 4.4 version 0.14.0

- BREAKING CHANGE: changed ‘mode’ to ‘modality’ throughout afmformats
- feat: introduced afmformats.AFMGroup, a container for AFMData (#11)
- feat: introduced afmformats.AFMQMap for managing quantitative AFMData (#11)
- feat: allow to use other derived classes of AFMData when loading experimental data via the *data\_classes\_by\_modality* option
- ref: *AFMData.export* is deprecated in favor of *AFMData.export\_data*
- ref: renamed submodule afm\_fdlist to mod\_force\_distance

## 4.5 version 0.13.3

- enh: improve speed when loading data by avoiding accessing data during initialization
- fix: JPK file format reader speed regression caused by #10 (implemented ArchiveCache)

## 4.6 version 0.13.2

- enh: make sure people don’t think they can load a data file with a different spring constant or sensitivity

## 4.7 version 0.13.1

- fix: make sure callback functions are always used

## 4.8 version 0.13.0

- feat: support zipped AFM workshop map data (#5)
- feat: added *find\_data* method
- enh: make MissingMetadataError class special (missing metadata are stored as property)
- enh: add “detect” function for JPK file format
- docs: add missing objects to `__all__` (autoapi)
- ref: code cleanup

## 4.9 version 0.12.6

- ref: DeprecationWarning: np.float from numpy 1.20

## 4.10 version 0.12.5

- fix: JPK file format reader kept the zip files open indefinitely which resulted in OSError “Too many open files” (#10)
- ci: removed appveyor build

## 4.11 version 0.12.4

- fix: opening .h5 files failed with AttributeError
- ref: setup.py test is deprecated

## 4.12 version 0.12.3

- build: migrate to GitHub Actions

## 4.13 version 0.12.2

- fix: properly sort curves within JPK files

## 4.14 version 0.12.1

- maintenance release

## 4.15 version 0.12.0

- ref: rewrite JPK data file reader: new JPKReader class (#4)
- enh: add new LazyData class for loading data on demand (#4)

## 4.16 version 0.11.0

- feat: allow defining “detect” method to determine whether a recipe can open a file (#7)

## 4.17 version 0.10.2

- maintenance release

## 4.18 version 0.10.1

- fix: parsing issue when loading .ibw files without AM/PM in “Time” metadata (#8)
- enh: make sure “time” is always parsed as HH:MM:SS.S when adding it to *MetaData*
- enh: compute “curve id” and “session id” from “date”, “time”, and “enum” if not given in *MetaData*

## 4.19 version 0.10.0

- feat: allow to override metadata when loading data
- feat: support new file format from AFM workshop (.csv)
- feat: support new file format from JPK (.jpk-qi-data)
- feat: support new file format from NT-MDT (.txt exported by Nova)
- enh: implement *AFMFormatRecipe* class for handling and verifying recipe dictionaries
- enh: implement *register\_format* function

## 4.20 version 0.9.0

- feat: support new file format from Asylum Research, Igor (.ibw)
- ref: always compute piezo range metadata instead of taking it from the set value in the acquisition settings (JKP format)

## 4.21 version 0.8.0

- enh: do not export “index” column to HDF5 files to save disk space
- enh: save column units when exporting to HDF5
- ref: moved class methods and constants from “afm\_fdist” to “afm\_data”
- docs: add code reference, basic usage, and list of file formats

## 4.22 version 0.7.1

- fix: exporting to HDF5 did not work when a `h5py.Group` was used
- fix: exporting to HDF5 did not reset the “enum” key
- enh: use gzip compression in HDF5 file format
- enh: allow “h5” and “hdf5” as HDF5 file format specifiers during export

## 4.23 version 0.7.0

- BREAKING CHANGE: piezo height is now loaded as “calibrated” and not as “nominal” (JPK file format)
- fix: metadata acquisition “duration” and “point count” only showed data of approach part (JPK file format)
- enh: load metadata “speed” and “rate” separately for approach and retract part
- ref: restructured meta data organization

## 4.24 version 0.6.0

- feat: force-distance metadata can now be saved and loaded for the .tab file format (#3)
- feat: implement new HDF5-based file format (read/write)
- feat: support piezo height (JPK file format)
- enh: improve parsing of JPK files (#1)

## 4.25 version 0.5.2

- ref: drop pandas dependency (#2)

## 4.26 version 0.5.1

- fix: allow “force-modulation” feedback mode

## 4.27 version 0.5.0

- feat: meta data summary with *MetaData.get\_summary*

## 4.28 version 0.4.1

- ref: group meta data by topic
- fix: identifier in JPK file format was actually session identifier

## 4.29 version 0.4.0

- BREAKING CHANGE: change metadata key names
- enh: add class for checking metadata

## 4.30 version 0.3.0

- feat: support tab-separated values file format (.tab)
- fix: file formats were not registered correctly
- ref: derive file format errors from own error classes

## 4.31 version 0.2.0

- compatibility changes towards nanite

## 4.32 version 0.1.0

- initial version



**BILBLIOGRAPHY**



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## BIBLIOGRAPHY

- [EBKS15] Yu.M. Efremov, D.V. Bagrov, M.P. Kirpichnikov, and K.V. Shaitan. Application of the johnson–kendall–roberts model in AFM-based mechanical measurements on cells and gel. *Colloids and Surfaces B: Biointerfaces*, 134:131–139, oct 2015. doi:[10.1016/j.colsurfb.2015.06.044](https://doi.org/10.1016/j.colsurfb.2015.06.044).
- [Efr20] Yuri Efremov. From fig. s6a, efremov, y.m., bagrov, d.v., kirpichnikov, m.p., shaitan, k.v., 2015. application of the johnson–kendall–roberts model in afm-based mechanical measurements on cells and gel. *figshare*, Feb 2020. doi:[10.6084/m9.figshare.11860053.v1](https://doi.org/10.6084/m9.figshare.11860053.v1).
- [HSC+18] Nicolas Hauck, Nalin Seixas, Silvia Centeno, Raimund Schlüßler, Gheorghe Cojoc, Paul Müller, Jochen Guck, Dominik Wöll, Ludger Wessjohann, and Julian Thiele. Droplet-assisted microfluidic fabrication and characterization of multifunctional polysaccharide microgels formed by multicomponent reactions. *Polymers*, 10(10):1055, sep 2018. doi:[10.3390/polym10101055](https://doi.org/10.3390/polym10101055).



## PYTHON MODULE INDEX

### a

- `afmformats`, 7
- `afmformats._version`, 7
- `afmformats._version_save`, 8
- `afmformats.afm_data`, 8
- `afmformats.afm_group`, 10
- `afmformats.afm_qmap`, 11
- `afmformats.errors`, 13
- `afmformats.fmt_hdf5`, 15
- `afmformats.fmt_igor`, 16
- `afmformats.fmt_jpk`, 17
- `afmformats.fmt_jpk.jpk_data`, 17
- `afmformats.fmt_jpk.jpk_meta`, 21
- `afmformats.fmt_jpk.jpk_reader`, 21
- `afmformats.fmt_ntmdt_txt`, 24
- `afmformats.fmt_tab`, 24
- `afmformats.fmt_workshop`, 25
- `afmformats.fmt_workshop.ws_map`, 25
- `afmformats.fmt_workshop.ws_single`, 26
- `afmformats.formats`, 27
- `afmformats.lazy_loader`, 30
- `afmformats.meta`, 31
- `afmformats.mod_force_distance`, 36
- `afmformats.parse_funcs`, 37





## A

AFMData (class in *afmformats.afm\_data*), 8  
 AFMFileFormatError (class in *afmformats.errors*), 13  
 AFMForceDistance (class in *afmformats.mod\_force\_distance*), 36  
 AFMFormatRecipe (class in *afmformats.formats*), 28  
 afmformats  
     module, 7  
 afmformats.\_version  
     module, 7  
 afmformats.\_version\_save  
     module, 8  
 afmformats.afm\_data  
     module, 8  
 afmformats.afm\_group  
     module, 10  
 afmformats.afm\_qmap  
     module, 11  
 afmformats.errors  
     module, 13  
 afmformats.fmt\_hdf5  
     module, 15  
 afmformats.fmt\_igor  
     module, 16  
 afmformats.fmt\_jpk  
     module, 17  
 afmformats.fmt\_jpk.jpk\_data  
     module, 17  
 afmformats.fmt\_jpk.jpk\_meta  
     module, 21  
 afmformats.fmt\_jpk.jpk\_reader  
     module, 21  
 afmformats.fmt\_ntmdt\_txt  
     module, 24  
 afmformats.fmt\_tab  
     module, 24  
 afmformats.fmt\_workshop  
     module, 25  
 afmformats.fmt\_workshop.ws\_map  
     module, 25  
 afmformats.fmt\_workshop.ws\_single  
     module, 26

afmformats.formats  
     module, 27  
 afmformats.lazy\_loader  
     module, 30  
 afmformats.meta  
     module, 31  
 afmformats.mod\_force\_distance  
     module, 36  
 afmformats.parse\_funcs  
     module, 37  
 AFMGroup (class in *afmformats.afm\_group*), 10  
 AFMQMap (class in *afmformats.afm\_qmap*), 12  
 append() (*afmformats.afm\_group.AFMGroup* method), 11  
 appr (*afmformats.mod\_force\_distance.AFMForceDistance* property), 36  
 ArchiveCache (class in *afmformats.fmt\_jpk.jpk\_reader*), 22  
 as\_dict() (*afmformats.meta.MetaData* method), 32

## C

column\_dtypes (in module *afmformats.afm\_data*), 9  
 column\_units (in module *afmformats.afm\_data*), 9  
 columns (*afmformats.afm\_data.AFMData* property), 9  
 copy() (*afmformats.meta.MetaData* method), 32

## D

DataFileBrokenError (class in *afmformats.errors*), 14  
 DEF\_ALL (in module *afmformats.meta*), 34  
 default\_data\_classes\_by\_modality (in module *afmformats.formats*), 29  
 descr (*afmformats.formats.AFMFormatRecipe* property), 28  
 detect() (*afmformats.formats.AFMFormatRecipe* method), 28

## E

enum (*afmformats.afm\_data.AFMData* property), 9  
 export\_data() (*afmformats.afm\_data.AFMData* method), 8  
 extent (*afmformats.afm\_qmap.AFMQMap* property), 13

## F

`fbool()` (in module `afmformats.parse_funcs`), 37  
`feat_core_data_height_base_point_um()` (*afmformats.afm\_qmap.AFMQMap* static method), 12  
`feat_core_data_piezo_range_um()` (*afmformats.afm\_qmap.AFMQMap* static method), 12  
`feat_core_data_scan_order()` (*afmformats.afm\_qmap.AFMQMap* static method), 12  
`features` (*afmformats.afm\_qmap.AFMQMap* attribute), 13  
`FileFormatMetadataError` (class in *afmformats.errors*), 14  
`FileFormatNotSupportedError` (class in *afmformats.errors*), 14  
`files` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* property), 23  
`find_column_dat()` (in module *afmformats.fmt\_jpk.jpk\_data*), 17  
`find_data()` (in module *afmformats.formats*), 27  
`fint()` (in module *afmformats.parse\_funcs*), 37  
`formats_available` (in module *afmformats.formats*), 29  
`formats_by_modality` (in module *afmformats.formats*), 29  
`formats_by_suffix` (in module *afmformats.formats*), 29

## G

`get()` (*afmformats.fmt\_jpk.jpk\_reader.ArchiveCache* static method), 22  
`get()` (*afmformats.meta.Metadata* method), 32  
`get_coords()` (*afmformats.afm\_qmap.AFMQMap* method), 12  
`get_data()` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* method), 22  
`get_enum()` (*afmformats.afm\_group.AFMGroup* method), 11  
`get_index_numbers()` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* method), 23  
`get_index_path()` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* method), 23  
`get_index_segment_numbers()` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* method), 23  
`get_index_segment_path()` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* method), 23  
`get_metadata()` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* method), 23

`get_primary_meta_recipe` (in module *afmformats.fmt\_jpk.jpk\_meta*), 21  
`get_qmap()` (*afmformats.afm\_qmap.AFMQMap* method), 12  
`get_recipe()` (in module *afmformats.formats*), 27  
`get_secondary_meta_recipe` (in module *afmformats.fmt\_jpk.jpk\_meta*), 21  
`get_summary()` (*afmformats.meta.Metadata* method), 32  
`group` (*afmformats.afm\_qmap.AFMQMap* attribute), 13

## H

`H5DictReader` (class in *afmformats.fmt\_hdf5*), 16  
`hierarchy` (*afmformats.fmt\_jpk.jpk\_reader.JPKReader* property), 23

## I

`InvalidFileFormatError` (class in *afmformats.errors*), 14  
`items()` (*afmformats.meta.Metadata* method), 32

## J

`JPK_COLUMNS` (in module *afmformats.fmt\_jpk.jpk\_data*), 20  
`JPK_SLOTS` (in module *afmformats.fmt\_jpk.jpk\_data*), 20  
`JPK_UNITS` (in module *afmformats.fmt\_jpk.jpk\_data*), 20  
`JPKReader` (class in *afmformats.fmt\_jpk.jpk\_reader*), 22

## K

`KEYS_VALID` (in module *afmformats.meta*), 35  
`known_columns` (in module *afmformats.afm\_data*), 10

## L

`LazyData` (class in *afmformats.lazy\_loader*), 30  
`load_csv()` (in module *afmformats.fmt\_workshop.ws\_single*), 26  
`load_dat_raw()` (in module *afmformats.fmt\_jpk.jpk\_data*), 18  
`load_dat_unit()` (in module *afmformats.fmt\_jpk.jpk\_data*), 18  
`load_data()` (in module *afmformats.formats*), 27  
`load_hdf5()` (in module *afmformats.fmt\_hdf5*), 15  
`load_igor()` (in module *afmformats.fmt\_igor*), 17  
`load_jpk()` (in module *afmformats.fmt\_jpk*), 23  
`load_map()` (in module *afmformats.fmt\_workshop.ws\_map*), 25  
`load_tab()` (in module *afmformats.fmt\_tab*), 24  
`load_txt()` (in module *afmformats.fmt\_ntmdt\_txt*), 24  
`loader` (*afmformats.formats.AFMFormatRecipe* property), 28

## M

`maker` (*afmformats.formats.AFMFormatRecipe* property), 28

META\_FIELDS (in module *afmformats.meta*), 32  
 meta\_keys (*afmformats.errors.MissingMetadataError* attribute), 15  
 metadata (*afmformats.afm\_data.AFMDData* property), 9  
 Metadata (class in *afmformats.meta*), 31  
 MetadataMissingError (class in *afmformats.meta*), 31  
 MissingMetadataError (class in *afmformats.errors*), 15  
 modality (*afmformats.afm\_data.AFMDData* property), 9  
 modality (*afmformats.formats.AFMFormatRecipe* property), 28  
 modality (*afmformats.mod\_force\_distance.AFMForceDistance* property), 36  
 module  
     afmformats, 7  
     afmformats.\_version, 7  
     afmformats.\_version\_save, 8  
     afmformats.afm\_data, 8  
     afmformats.afm\_group, 10  
     afmformats.afm\_qmap, 11  
     afmformats.errors, 13  
     afmformats.fmt\_hdf5, 15  
     afmformats.fmt\_igor, 16  
     afmformats.fmt\_jpk, 17  
     afmformats.fmt\_jpk.jpk\_data, 17  
     afmformats.fmt\_jpk.jpk\_meta, 21  
     afmformats.fmt\_jpk.jpk\_reader, 21  
     afmformats.fmt\_ntmdt\_txt, 24  
     afmformats.fmt\_tab, 24  
     afmformats.fmt\_workshop, 25  
     afmformats.fmt\_workshop.ws\_map, 25  
     afmformats.fmt\_workshop.ws\_single, 26  
     afmformats.formats, 27  
     afmformats.lazy\_loader, 30  
     afmformats.meta, 31  
     afmformats.mod\_force\_distance, 36  
     afmformats.parse\_funcs, 37

## P

parse\_time() (in module *afmformats.meta*), 31  
 path (*afmformats.afm\_data.AFMDData* property), 9

## Q

qmap\_feature() (in module *afmformats.afm\_qmap*), 11

## R

ReadJPKErrors (class in *afmformats.fmt\_jpk.jpk\_data*), 20  
 ReadJPKMetaKeyError (class in *afmformats.fmt\_jpk.jpk\_meta*), 21  
 retri (*afmformats.mod\_force\_distance.AFMForceDistance* property), 36

## S

Segment (class in *afmformats.mod\_force\_distance*), 36  
 set\_lazy\_loader() (*afmformats.lazy\_loader.LazyData* method), 30  
 shape (*afmformats.afm\_qmap.AFMQMap* property), 13  
 subgroup\_with\_path() (*afmformats.afm\_group.AFMGroup* method), 11  
 suffix (*afmformats.formats.AFMFormatRecipe* property), 28  
 supported\_extensions (in module *afmformats.formats*), 30

## U

unit\_scales (in module *afmformats.afm\_qmap*), 13  
 update() (*afmformats.meta.Metadata* method), 32

## V

values() (*afmformats.meta.Metadata* method), 32  
 vd\_str\_in() (in module *afmformats.parse\_funcs*), 37

## W

which (*afmformats.mod\_force\_distance.Segment* attribute), 37