
afmformats Documentation

Release 0.18.7

Paul Müller

Apr 09, 2026

CONTENTS

1	Getting started	3
1.1	Installation	3
1.2	Basic Usage	3
1.3	Supported file formats	3
1.4	Notes	5
2	Examples	7
2.1	Double-exponential fit to stress-relaxation data	7
3	Advanced Usage	11
3.1	Grouping AFM data	11
3.2	Logging (for developers)	12
4	Implementing new file formats	13
4.1	Basic file format reader structure	13
4.2	Optimizing data import	15
5	afmformats	17
5.1	Submodules	17
5.1.1	afmformats._version	17
5.1.2	afmformats.afm_data	18
5.1.3	afmformats.afm_group	20
5.1.4	afmformats.afm_qmap	21
5.1.5	afmformats.afm_segment	23
5.1.6	afmformats.errors	24
5.1.7	afmformats.formats	25
5.1.8	afmformats.lazy_loader	40
5.1.9	afmformats.logging_setup	40
5.1.10	afmformats.meta	41
5.1.11	afmformats.mod_creep_compliance	46
5.1.12	afmformats.mod_force_distance	47
5.1.13	afmformats.mod_stress_relaxation	48
5.1.14	afmformats.parse_funcs	49
6	Changelog	51
6.1	version 0.18.7	51
6.2	version 0.18.6	51
6.3	version 0.18.5	51
6.4	version 0.18.4	51
6.5	version 0.18.3	51
6.6	version 0.18.2	51

6.7	version 0.18.1	51
6.8	version 0.18.0	52
6.9	version 0.17.1	52
6.10	version 0.17.0	52
6.11	version 0.16.7	52
6.12	version 0.16.6	52
6.13	version 0.16.5	52
6.14	version 0.16.4	52
6.15	version 0.16.3	52
6.16	version 0.16.2	52
6.17	version 0.16.1	53
6.18	version 0.16.0	53
6.19	version 0.15.0	53
6.20	version 0.14.4	53
6.21	version 0.14.3	53
6.22	version 0.14.2	54
6.23	version 0.14.1	54
6.24	version 0.14.0	54
6.25	version 0.13.3	54
6.26	version 0.13.2	54
6.27	version 0.13.1	54
6.28	version 0.13.0	55
6.29	version 0.12.6	55
6.30	version 0.12.5	55
6.31	version 0.12.4	55
6.32	version 0.12.3	55
6.33	version 0.12.2	55
6.34	version 0.12.1	55
6.35	version 0.12.0	55
6.36	version 0.11.0	56
6.37	version 0.10.2	56
6.38	version 0.10.1	56
6.39	version 0.10.0	56
6.40	version 0.9.0	56
6.41	version 0.8.0	56
6.42	version 0.7.1	56
6.43	version 0.7.0	57
6.44	version 0.6.0	57
6.45	version 0.5.2	57
6.46	version 0.5.1	57
6.47	version 0.5.0	57
6.48	version 0.4.1	57
6.49	version 0.4.0	57
6.50	version 0.3.0	57
6.51	version 0.2.0	58
6.52	version 0.1.0	58
7	Bibliography	59
8	Indices and tables	61
	Bibliography	63
	Python Module Index	65

Afmformats is a Python library for reading atomic force microscopy (AFM) data file formats. This is the documentation of afmformats version 0.18.7.

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 0x7b8c2f3e12d0>]

# 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], shape=(8400,))
```

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	com sepa val- ues	.csv	<i>formats.fmt_workshop.ws_single.load_csv</i>
afmformats	HDF base	.h5	<i>formats.fmt_hdf5.load_hdf5</i>
Asylum Research	bi- na- ry- wave	.ibw	<i>formats.fmt_igor.load_igor</i>
JPK Instruments	bi- nary FD data	.jpk-force	<i>formats.fmt_jpk.load_jpk</i>
JPK Instruments	bi- nary QM; data	.jpk-force-map	<i>formats.fmt_jpk.load_jpk</i>
JPK Instruments	bi- nary QM; data	.jpk-qi-data	<i>formats.fmt_jpk.load_jpk</i>
JPK Instruments	bi- nary QM; data	.jpk-qi-series	<i>formats.fmt_jpk.load_jpk</i>
afmformats	tab- sepa val- ues	.tab	<i>formats.fmt_tab.load_tab</i>
NT-MDT Spectrum Instruments	ex- port by NT- MDT Nov	.txt	<i>formats.fmt_ntmdt_txt.load_txt</i>
Optics11 Life	ex- port by Op- tics1 Chia In- den- ter	.txt	<i>formats.fmt_chiara_txt.load_txt</i>
AFM work- shop	QM. as zipp com	.zip	<i>formats.fmt_workshop.ws_map.load_map</i>
4	sepa val- ues		Chapter 1. Getting started

1.4 Notes

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

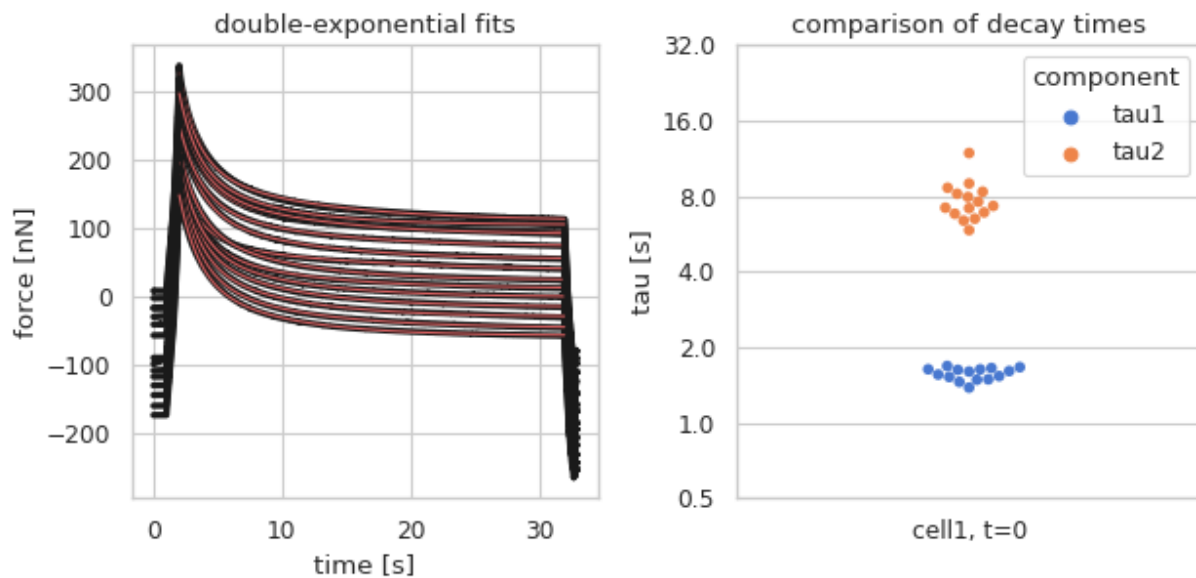
EXAMPLES

These are examples for the bare `afmformats` package. You might find other examples interesting to you in the [nanite documentation](#).

2.1 Double-exponential fit to stress-relaxation data

This example reproduces the first entry in figure 5d (cell1) of [BAZ+22].

Please note that in the original manuscript, not all curves were used in the final figure. Some curves were excluded based on curve quality. The data were kindly provided by Alice Battistella.



stress_relaxation.py

```

1 import pathlib
2 import shutil
3 import tempfile
4
5 import afmformats
6
7 from lmfit.models import ConstantModel, ExponentialModel
8 import matplotlib.pyplot as plt

```

(continues on next page)

(continued from previous page)

```
9 from matplotlib.ticker import ScalarFormatter
10 import numpy as np
11 import pandas
12 import seaborn as sns
13 sns.set_theme(style="whitegrid", palette="muted")
14
15
16 # extract the data to a temporary directory
17 data_path = pathlib.Path(__file__).parent / "data"
18 wd_path = pathlib.Path(tempfile.mkdtemp())
19 shutil.unpack_archive(data_path / "10.1002_btm2.10294_fig5d-cell1.zip",
20                      wd_path)
21
22 # scale conversion constants
23 xsc = 1
24 ysc = 1e9
25 xl = "time [s]"
26 yl = "force [nN]"
27
28 # data extraction
29 data = []
30 for path in wd_path.glob("*.jpk-force"):
31     data += afmformats.load_data(path)
32
33 # this is where the data for the swarm plot is stored
34 rdat = {
35     "tau": [],
36     "component": [],
37     "cell": [],
38 }
39
40 data_fits = []
41
42 for di in data:
43     # use intermediate `intr` segment data
44     x = di.intr["time"] * xsc
45     y = di.intr["force"] * ysc
46
47     # fitting with double-exponential
48     const = ConstantModel()
49     exp_1 = ExponentialModel(prefix='exp1_')
50     exp_2 = ExponentialModel(prefix='exp2_')
51
52     pars = exp_1.guess(y, x=x)
53     pars.update(exp_2.guess(y, x=x))
54     pars["exp1_decay"].set(value=1, min=0.1)
55     pars["exp2_decay"].set(value=8, min=2)
56     pars.update(const.guess(y, x=x))
57     pars["c"].set(value=np.min(y))
58
59     mod = const + exp_1 + exp_2
60
```

(continues on next page)

(continued from previous page)

```
61 init = mod.eval(pars, x=x)
62 out = mod.fit(y, pars, x=x)
63 data_fits.append([x, out.best_fit])
64
65 rdat["tau"].append(out.params["exp1_decay"].value)
66 rdat["component"].append("tau1")
67 rdat["cell"].append("cell1, t=0")
68
69 rdat["tau"].append(out.params["exp2_decay"].value)
70 rdat["component"].append("tau2")
71 rdat["cell"].append("cell1, t=0")
72
73
74 fig = plt.figure(figsize=(8, 4))
75
76 # plot all stress-relaxation curves
77 ax1 = plt.subplot(121, title="double-exponential fits")
78 for ii in range(len(data)):
79     di = data[ii]
80     ax1.plot(di["time"]*xsc, di["force"]*ysc, color="k", lw=3)
81     xf, yf = data_fits[ii]
82     ax1.plot(xf, yf, color="r", lw=1)
83 ax1.set_xlabel(xl)
84 ax1.set_ylabel(yl)
85
86 df = pandas.DataFrame(data=rdat)
87 # Draw a categorical scatterplot to show each observation
88 ax2 = plt.subplot(122, title="comparison of decay times")
89 ax2 = sns.swarmplot(data=df, x="cell", y="tau", hue="component")
90 ax2.set_yscale('log', base=2)
91 ax2.yaxis.set_major_formatter(ScalarFormatter())
92 ax2.set_yticks([0.5, 1, 2, 4, 8, 16, 32])
93 ax2.set(xlabel="", ylabel="tau [s]")
94
95 plt.tight_layout()
96
97 plt.show()
```


ADVANCED USAGE

3.1 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]
```

3.2 Logging (for developers)

afmformats has a simple logging system. When loading data in a script or in the console, debug-level logs can be written to *afmformats.log* in your machine's temp folder if you call `afmformats.configure_logging()`. When running tests via `pytest`, the logs will be written to a temp directory as defined in `tests.conftest.pytest_configure()`.

```
import pathlib
import afmformats

data_path = pathlib.Path("tests/data")

# write logs to tmp/afmformats.log
afmformats.configure_logging()

afmformats.load_data(data_path / "fmt-hdf5-fd_version_0.13.3.h5")
```

If you would like the logs also to be output to the terminal when running scripts you can set the logging level:

```
import pathlib
import afmformats
import logging

data_path = pathlib.Path("tests/data")

# write logs to tmp/afmformats.log and to terminal
afmformats.configure_logging(console_logging_level=logging.DEBUG)

afmformats.load_data(data_path / "fmt-hdf5-fd_version_0.13.3.h5")
```

Output of the above script:

```
DEBUG:afmformats.formats:Loaded 1 dataset(s) from '...\afmformats\tests\data\fmt-hdf5-fd_
↪version_0.13.3.h5' using 'HDF5-based'
Process finished with exit code 0
```

IMPLEMENTING NEW FILE FORMATS

If you are interested in adding support for a new file format, please [create a new issue](#) to start a discussion. Please also attach a zip file with example data that can later on be used during testing.

If you are familiar with GitHub, please create a pull request and make sure that

- the file format reader is located in `afmformats.formats.fmt_NAME` (it may be a directory or a file, depending on the complexity)
- the file format displays correctly [in the docs](#) and the docs compile without errors:

```
cd docs
pip install -r requirements.txt
sphinx-build . _build
# and open _build/index.html in a browser
```

- you updated the CHANGELOG
- your code is fully tested (create test functions in `tests/test_fmt_NAME.py`) and all other tests pass (There are a few general tests that all file format readers must pass):

```
pip install pytest
pytest tests
```

- the data files for examples are named according to `fmt-NAME-MOD_filename.suffix` where MOD can be e.g. fd for force-distance data.

If you cannot or will not work with GitHub, you may paste your code in the corresponding issue. If the file format is not too complicated, let's just hope that things don't get messy.

4.1 Basic file format reader structure

The best way to understand how file formats work in `afmformats` is to take a look at the [file formats implemented already](#). For the sake of clarity, here is a `file format reader template`:

```
import pathlib

import numpy as np

__all__ = ["load_my_format"]

def load_my_format(path, callback=None, meta_override=None):
```

(continues on next page)

```

"""Loads AFM data from my format

This is the main function for loading your file format. Please
add a description here.

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 :data:`afmformats.meta.META_FIELDS`)
"""
if meta_override is None:
    meta_override = {}

path = pathlib.Path(path)
# Here you would start parsing your data and metadata from `path`
# You should specify as many metadata keys as possible. See
# afmformats.meta.DEF_ALL for a list of valid keys.
metadata = {"path": path}
# Valid column names are defined in afmformats.afm_data.known_columns.
data = {"force": np.linspace(1e-9, 5e-9, 100),
        "height (measured)": np.linspace(2e-6, -1e-6, 100)}

metadata.update(meta_override)
dd = {"data": data,
      "metadata": metadata}

if callback is not None:
    callback(1)

# You may also return a list with more items in case the file format
# contains more than one curve.
return [dd]

recipe_myf = {
    "descr": "A short description",
    "loader": load_my_format,
    "suffix": ".myf",
    "modality": "force-distance",
    "maker": "designer of file format",
}

```

A few notes:

- The `recipe_myf` contains the recipe for loading the file format into `afmformats`. It must be registered in `afmformats/formats/__init__.py`.

- You may call the `callback` function with a floating point value between 0 and 1 (progress tracking) in-between of your loading steps if you expect that your file format reader is slow (e.g. several curves have to be loaded). This will give users of e.g. PyJibe visual feedback on how long they will have to wait.
- The `meta_override` dictionary is useful if your file format does not contain essential metadata such as spring constant or sensitivity. In such cases, you can raise an `afmformats.errors.MissingMetadataError` to signal PyJibe that it should ask the user for the missing metadata. For an example, please see the AFM workshop file format.

4.2 Optimizing data import

In most cases, it is not necessary to actually load the data from disk in the `load_my_format` method, especially if you have to parse large binary blobs or text files. In such cases, you can make use of the `lazy loaders` implemented in `afmformats`. For metadata, you can use `afmformats.meta.LazyMetaValue` and for data columns, you can use `afmformats.lazy_loader.LazyData`. The JPK file reader makes heavy usage of those classes.

- *Submodules*

5.1 Submodules

5.1.1 afmformats._version

- *Variables*

Variables

- `__version__`
- `__version_tuple__`
- `version`
- `version_tuple`
- `__commit_id__`
- `commit_id`

`afmformats._version.__version__`

`str(object=)` -> `str` `str(bytes_or_buffer[, encoding[, errors]])` -> `str`

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. `encoding` defaults to `sys.getdefaultencoding()`. `errors` defaults to 'strict'.

```
'0.18.7'
```

`afmformats._version.__version_tuple__`

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

```
(0, 18, 7)
```

afmformats._version.version

```
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

```
'0.18.7'
```

afmformats._version.version_tuple

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

```
(0, 18, 7)
```

afmformats._version.__commit_id__

```
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

```
'g731b5ad88'
```

afmformats._version.commit_id

```
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

```
'g731b5ad88'
```

5.1.2 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

reset_data()

Resets all data to the state they were after loading

Internally, only *self._data* is *clear`ed*, which means that all calls to `__getitem__` fall-back to *self._raw_data*.

property columns

Available data columns

property columns_innate

Data columns available only in the original data file

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 'numpy.uint8'>,
 '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']
```

5.1.3 afmformats.afm_group

- *Classes*

Classes

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

class `afmformats.afm_group.AFMGroup`(*path=None, meta_override=None, callback=None, modality=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 *index* to pass *Indentation* (which is a subclass of the default *AFMForceDistance*) for handling “force-indentation” data.

Inheritance

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*

5.1.4 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, modality=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 `index` to pass `Indentation` (which is a subclass of the default `AFMForceDistance`) for handling “force-indentation” data.

Inheritance

`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) [μm]

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`

Scale conversion helper

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

5.1.5 afmformats.afm_segment

- *Classes*

Classes

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

class `afmformats.afm_segment.AFMSegment`(*raw_data*, *data*, *segment*)

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

This class also caches the segment indices.

New Segment data

Parameters

- **raw_data** (*dict*) – dictionary containing valid column names as keys and 1d ndarrays as values; this is raw data (e.g. from the measurement file) that may be lazily-loaded

- **data** (*dict*) – same as `raw_data`, but in this case the data are already in memory; we distinguish between `raw_data` and `data` so that we know where the data came from (e.g. there might be “tip position” in both dictionaries, but we only always use (and override) the “tip position” in `data`. We never touch `raw_data`.

Inheritance

`clear_cache()`

Invalidates the segment indices corresponding to `self.data`

segment

The segment type (approach, intermediate, or retract)

property segment_indices

boolean array of segment indices

5.1.6 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

```
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

5.1.7 afmformats.formats

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

Submodules

`afmformats.formats.fmt_chiario_txt`

- *Functions*

Functions

- `load_txt()`: Load text files exported by the Optics11 Chiaro Indenter.

`afmformats.formats.fmt_chiario_txt.load_txt` (*path*, *callback=None*, *meta_override=None*)

Load text files exported by the Optics11 Chiaro Indenter.

Parameters

- **path** (*str* or *pathlib.Path* or *io.TextIOBase*) – path to an chiaro-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`)

`afmformats.formats.fmt_hdf5`

- *Functions*
- *Classes*

Functions

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

`afmformats.formats.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.formats.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

`afmformats.formats.fmt_igor`

- *Functions*

Functions

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

`afmformats.formats.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`)

`afmformats.formats.fmt_jpk`

- *Submodules*
- *Functions*

Submodules`afmformats.formats.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.formats.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.formats.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.formats.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”.

```
distance = 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.formats.fmt_jpk.jpk_data.ReadJPKError
```

Inheritance

Variables

- *JPK_COLUMNS*
- *JPK_SLOTS*
- *JPK_UNITS*

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

Maps afmformats column names to default JPK normalization slots

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

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

Maps afmformats column names to default JPK units

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

`afmformats.formats.fmt_jpk.jpk_meta`

- *Variables*

Variables

- `get_primary_meta_recipe`
- `get_secondary_meta_recipe`

`afmformats.formats.fmt_jpk.jpk_meta.get_primary_meta_recipe`

```
<functools._lru_cache_wrapper object at 0x7a7d579b64b0>
```

`afmformats.formats.fmt_jpk.jpk_meta.get_secondary_meta_recipe`

```
<functools._lru_cache_wrapper object at 0x7a7d579b6560>
```

`afmformats.formats.fmt_jpk.jpk_reader`

- *Classes*

Classes

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

class `afmformats.formats.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

static `get(zip_path)`

Return the (possibly cached) *ZipFile* object for `zip_path`

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

Inheritance

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.

set_metadata(*metadata*)

Override internal metadata

This has a direct effect on `get_metadata()`.

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.formats.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`)

`afmformats.formats.fmt_ntmdt_txt`

- *Functions*

Functions

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

`afmformats.formats.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/), 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`)

`afmformats.formats.fmt_tab`

- *Functions*

Functions

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

`afmformats.formats.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`)

`afmformats.formats.fmt_workshop`

- *Submodules*

Submodules

`afmformats.formats.fmt_workshop.ws_map`

- *Functions*

Functions

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

`afmformats.formats.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
```

(continues on next page)

(continued from previous page)

```

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.formats.fmt_workshop.ws_single

- *Functions*

Functions

- `load_csv()`: Load csv data from AFM workshop

```
afmformats.formats.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)

```

(continues on next page)

(continued from previous page)

```

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

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 (see *IMAGING_MODALITIES*)

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 (e.g. “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 *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

```
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.

```
get_modality(path)
```

Determine modality of a path

If a recipe provides several modalities, load the dataset and get the modality from the metadata.

```
property descr
```

description of file format

property loader

method for loading the data

property maker

who introduced the file format

property modalities

list of supported AFM imaging modalities

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

```
{'creep-compliance': <class 'afmformats.mod_creep_compliance.AFMCreepCompliance'>,
 'force-distance': <class 'afmformats.mod_force_distance.AFMForceDistance'>,
 'stress-relaxation': <class 'afmformats.mod_stress_relaxation.AFMStressRelaxation'>
 ↪}
```

afmformats.formats.formats_available

available/supported file formats

```
[<AFMFormatRecipe from 'afmformats.formats.fmt_hdf5' at 0x7a7d575bc610>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_igor' at 0x7a7d575bc690>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at 0x7a7d575bc710>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at 0x7a7d575bc810>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at 0x7a7d575bc890>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at 0x7a7d575bc9d0>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_ntmdt_txt' at 0x7a7d575bca50>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_chiaro_txt' at 0x7a7d575bcd0>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_tab' at 0x7a7d575bcb10>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_workshop.ws_map' at 0x7a7d575bc950>,
 <AFMFormatRecipe from 'afmformats.formats.fmt_workshop.ws_single' at
 ↪0x7a7d575bc990>]
```

afmformats.formats.formats_by_suffix

available file formats in a dictionary with suffix keys

```
{'.csv': [<AFMFormatRecipe from 'afmformats.formats.fmt_workshop.ws_single' at
 ↪0x7a7d575bc990>],
 '.h5': [<AFMFormatRecipe from 'afmformats.formats.fmt_hdf5' at 0x7a7d575bc610>],
 '.ibw': [<AFMFormatRecipe from 'afmformats.formats.fmt_igor' at 0x7a7d575bc690>],
 '.jpk-force': [<AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at
```

(continues on next page)

(continued from previous page)

```

↪0x7a7d575bc710>],
'.jpk-force-map': [<AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc810>],
'.jpk-qi-data': [<AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc890>],
'.jpk-qi-series': [<AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc9d0>],
'.tab': [<AFMFormatRecipe from 'afmformats.formats.fmt_tab' at 0x7a7d575bcb10>],
'.txt': [<AFMFormatRecipe from 'afmformats.formats.fmt_ntmdt_txt' at_
↪0x7a7d575bca50>,
      <AFMFormatRecipe from 'afmformats.formats.fmt_chiaro_txt' at_
↪0x7a7d575bcad0>],
'.zip': [<AFMFormatRecipe from 'afmformats.formats.fmt_workshop.ws_map' at_
↪0x7a7d575bc950>]]

```

afmformats.formats.formats_by_modality

available file formats in a dictionary for each modality

```

{'creep-compliance': [<AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc710>,
                    <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc810>,
                    <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc890>,
                    <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc9d0>],
'force-distance': [<AFMFormatRecipe from 'afmformats.formats.fmt_hdf5' at_
↪0x7a7d575bc610>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_igor' at_
↪0x7a7d575bc690>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc710>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc810>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc890>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc9d0>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_ntmdt_txt' at_
↪0x7a7d575bca50>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_chiaro_txt' at_
↪0x7a7d575bcad0>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_tab' at_
↪0x7a7d575bcb10>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_workshop.ws_map'_
↪at 0x7a7d575bc950>,
                  <AFMFormatRecipe from 'afmformats.formats.fmt_workshop.ws_single
↪' at 0x7a7d575bc990>],
'stress-relaxation': [<AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc710>,
                    <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at_
↪0x7a7d575bc810>,

```

(continues on next page)

(continued from previous page)

```

    <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at
    ↪0x7a7d575bc890>,
    <AFMFormatRecipe from 'afmformats.formats.fmt_jpk' at
    ↪0x7a7d575bc9d0>]}

```

afmformats.formats.supported_extensions

list of supported extensions

```

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

```

5.1.8 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**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

5.1.9 afmformats.logging_setup

Package-wide logging configuration.

- *Functions*
- *Variables*

Functions

- `configure_logging()`: Ensure afmformats writes debug logs to a file.

```
afmformats.logging_setup.configure_logging(log_path: str | None = '/tmp/afmformats.log',
                                          console_logging_level: bool | int = 0) → str
```

Ensure afmformats writes debug logs to a file.

Returns the log path used so CI pipelines can publish it as an artifact.

Variables

- `DEFAULT_LOG_PATH`

```
afmformats.logging_setup.DEFAULT_LOG_PATH
```

```
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

```
'/tmp/afmformats.log'
```

5.1.10 afmformats.meta

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

Functions

- `parse_time()`: Convert a time string to “HH:MM:SS.S”

```
afmformats.meta.parse_time(time_str: str)
```

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
- *LazyMetaValue*: A metadata value that is evaluated lazily in *MetaData*
- *MetaData*: Management of meta data variables

class afmformats.meta.MetaDataMissingError

Raised when meta data is missing

Inheritance

class afmformats.meta.LazyMetaValue(func, *args, **kwargs)

A metadata value that is evaluated lazily in *MetaData*

Example usage:

```
meta = afmformats.meta.MetaData
meta["z range"] = afmformats.meta.LazyMetaValue(
    np.ptp,
    np.arange(10))
```

Parameters

- **func** (*callable*) – Function to call to get the metadata value
- **args** – arguments to **func**
- **kwargs** – Keyword arguments to **func**

Inheritance

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

Management of meta data variables

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

Inheritance

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(key, default=None)

Return the value for key if key is in the dictionary, else default.

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

- *IMAGING_MODALITIES*
- *META_FIELDS*
- *DEF_ALL*
- *KEYS_VALID*

afmformats.meta.IMAGING_MODALITIES

supported imaging modalities

```
['creep-compliance', 'force-distance', 'stress-relaxation']
```

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_
↪0x7a7d582a51c0>],
                 'imaging mode': ['Imaging modality',
                                   '',
                                   <function vd_str_in.<locals>.str_in at_
↪0x7a7d582a5300>],
                 'sensitivity': ['Sensitivity', 'm/V', <class 'float'>],
                 'spring constant': ['Cantilever spring constant',
                                     'N/m',
                                     <class 'float'>}},
 'dataset': {'duration': ['Duration', 's', <class 'float'>],
             'duration approach': ['Duration of approach segment',
                                   's',
                                   <class 'float'>],
             'duration retract': ['Duration of retract segment',
                                  's',
                                  <class 'float'>],
             'enum': ['Dataset index within the experiment',
                     '',
                     <function fint at 0x7a7d582a5580>],
             'point count': ['Size of the dataset in points',
                             '',
                             <function fint at 0x7a7d582a5580>],
```

(continues on next page)

(continued from previous page)

```

'rate approach': ['Sampling rate of approach segment',
                  'Hz',
                  <class 'float'>],
'rate retract': ['Sampling rate of retract segment',
                  'Hz',
                  <class 'float'>],
'segment count': ['Number of segments',
                  '',
                  <function fint at 0x7a7d582a5580>],
'setpoint': ['Target indentation force', 'N', <class 'float'>],
'speed approach': ['Piezo speed of approach segment',
                   'm/s',
                   <class 'float'>],
'speed retract': ['Piezo speed of retract segment',
                   'm/s',
                   <class 'float'>],
'z range': ['Axial piezo range', 'm', <class 'float'>]],
'dataset-mod creep-compliance': {'duration intermediate': ['Duration of '
                                                           'intermediate '
                                                           'segment',
                                                           's',
                                                           <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 0x7a7d582a5580>],
         'grid index y': ['Vertical grid position index',
                           '',
                           <function fint at 0x7a7d582a5580>],
         'grid shape x': ['Horizontal grid shape',
                           'px',
                           <function fint at 0x7a7d582a5580>],
         'grid shape y': ['Vertical grid shape',
                           'px',
                           <function fint at 0x7a7d582a5580>],
         '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'>]],
'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'>],
'duration approach': ['Duration of approach segment', 's', <class 'float'>],
'duration intermediate': ['Duration of intermediate segment',
                          's',
                          <class 'float'>],
'duration retract': ['Duration of retract segment', 's', <class 'float'>],
'enum': ['Dataset index within the experiment',
         '',
         <function fint at 0x7a7d582a5580>],
'feedback mode': ['Feedback mode',
                  '',
                  <function vd_str_in.<locals>.str_in at 0x7a7d582a51c0>],
'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 0x7a7d582a5580>],
'grid index y': ['Vertical grid position index',
                 '',
                 <function fint at 0x7a7d582a5580>],
'grid shape x': ['Horizontal grid shape',
                 'px',
                 <function fint at 0x7a7d582a5580>],
'grid shape y': ['Vertical grid shape',
                 'px',
                 <function fint at 0x7a7d582a5580>],
'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 0x7a7d582a5300>],
'instrument': ['Instrument', '', <class 'str'>],
'path': ['Path', '', <class 'pathlib.Path'>],
'point count': ['Size of the dataset in points',
                '',
                <function fint at 0x7a7d582a5580>],
'position x': ['Horizontal position', 'm', <class 'float'>],
'position y': ['Vertical position', 'm', <class 'float'>],
'rate approach': ['Sampling rate of approach segment', 'Hz', <class 'float'>],
'rate retract': ['Sampling rate of retract segment', 'Hz', <class 'float'>],
'segment count': ['Number of segments', '', <function fint at 0x7a7d582a5580>],
'sensitivity': ['Sensitivity', 'm/V', <class 'float'>],
'session id': ['Dataset identifier', '', <class 'str'>],
'setpoint': ['Target indentation force', 'N', <class 'float'>],
'software': ['Acquisition software', '', <class 'str'>],
'software version': ['Acquisition software version', '', <class 'str'>],
'speed approach': ['Piezo speed of approach segment', 'm/s', <class 'float'>],
```

(continues on next page)

(continued from previous page)

```
'speed retract': ['Piezo speed of retract segment', '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',
'duration approach',
'duration intermediate',
'duration retract',
'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',
'segment count',
'sensitivity',
'session id',
'setpoint',
'software',
'software version',
'speed approach',
'speed retract',
'spring constant',
'time',
'z range']
```

5.1.11 afmformats.mod_creep_compliance

- *Classes*

Classes

- *AFMCreepCompliance*: Base class for AFM creep-compliance data

class afmformats.mod_creep_compliance.**AFMCreepCompliance**(*args, **kwargs)

Base class for AFM creep-compliance data

A creep-compliance dataset consists of an approach, an intermediate (with constant Force), and a retract curve.

Initialization

Parameters

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

Inheritance

appr

Dictionary-like interface to the approach segment

intr

Dictionary-like interface to the intermediate segment

property modality

Imaging modality

retr

Dictionary-like interface to the retract segment

5.1.12 afmformats.mod_force_distance

- *Classes*

Classes

- *AFMForceDistance*: Base class for AFM force-distance data

class afmformats.mod_force_distance.**AFMForceDistance**(*args, **kwargs)

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

appr

Dictionary-like interface to the approach segment

property modality

Imaging modality

retr

Dictionary-like interface to the retract segment

5.1.13 afmformats.mod_stress_relaxation

- *Classes*

Classes

- *AFMStressRelaxation*: Base class for AFM stress-relaxation data

class afmformats.mod_stress_relaxation.**AFMStressRelaxation**(*args, **kwargs)

Base class for AFM stress-relaxation data

A stress-relaxation dataset consists of a pre-approach (with constant height), an approach (with a predefined indentation depth), an intermediate (with constant height), and a retract curve.

Initialization

Parameters

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

Inheritance

appr

Dictionary-like interface to the approach segment

intr

Dictionary-like interface to the intermediate segment

property modality

Imaging modality

prep

Dictionary-like interface to the pre-measurement segment

retr

Dictionary-like interface to the retract segment

5.1.14 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.

6.1 version 0.18.7

- enh: add logging system (#30)
- ref: cleanup

6.2 version 0.18.6

- enh: remove escape sequences when parsing time (#43)

6.3 version 0.18.5

- enh: create chiaro text format loader (#38)
- setup: migrate to pyproject.toml package file (#39)

6.4 version 0.18.4

- fix: update np.loadtxt with old default encoding argument (#29)

6.5 version 0.18.3

- ci: fix readthedocs.yml

6.6 version 0.18.2

- docs: release limitations on docs requirements

6.7 version 0.18.1

- ref: Migrate from pkg_resources to importlib.resources (#26)

6.8 version 0.18.0

- ref: JPK calibration files are now not ignored anymore
- ref: allow to specify custom metadata for JPK files
- ref: replaced ReadJPKMetaKeyError with MissingMetaDataError
- ref: support *return_modality* in *detect* methods of recipes
- ref: cleanup of JPK reader class

6.9 version 0.17.1

- maintenance release

6.10 version 0.17.0

- feat: support loading stress-relaxation data from .jpk-force files (#9)
- docs: add example for stress-relaxation data analysis

6.11 version 0.16.7

- maintenance release

6.12 version 0.16.6

- setup: bump igor2 from 0.4.0 to 0.5.0

6.13 version 0.16.5

- setup: migrate from igor to igor2 (Python 3 support)
- tests: add test for files that contain force data but no spring constant

6.14 version 0.16.4

- enh: implement *AFMData.columns_innate* which is a list of columns present in the original data file
- enh: implement *AFMData.reset_data* to remove all user-defined column data (i.e. columns not in *AFMData.columns_innate* or columns that override those)

6.15 version 0.16.3

- fix: setting column data to *AFMSegment* did not work when column did only exist in raw data property

6.16 version 0.16.2

- enh: implement *__setitem__* for *AFMSegment*
- ref: make *data* and *raw_data* private properties of *AFMSegment*

6.17 version 0.16.1

- enh: support .jpk-qi-series files (by extension)
- enh: improve error handling when data exported from the JPK software does not contain encoder scaling offset and multiplier
- ref: use f-strings more often

6.18 version 0.16.0

- BREAKING CHANGE: “segment” column is no longer boolean, so it cannot be used directly for indexing (#15)
- fix: support modified AFM workshop file format (#17)
- fix: segment data via *AFMData.appr* and *AFMData.retr* did not use user-defined data to identify segments, but the “segment” column from the raw data (this does not affect PyJibe or nanite which fo not make excessive use of these properties)
- enh: add “segment count” key to meta data
- enh: optimize segment handling: new class *AFMSegment* that caches segment indices (#16)
- enh: properly test data types of metadata when setting them

6.19 version 0.15.0

- feat: generally support creep-compliance and stress-relaxation data via the new “modalities” (supercedes “modality”) recipe key
- feat: support for JPK creep-compliance data
- enh: extract segment duration from JPK files
- fix: ignore NaN values in JPK property files
- ref: add separate meta data section for modality-related keys
- ref: improvements of JPK metadata reader
- tests: rename test data files to reflect format

6.20 version 0.14.4

- docs: add section for implementing new file formats (#13)
- fix: *IndexError* when trying to open .txt files that are no AFM files
- ref: separate submodule for formats

6.21 version 0.14.3

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

6.22 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

6.23 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

6.24 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_fdist* to *mod_force_distance*

6.25 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)

6.26 version 0.13.2

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

6.27 version 0.13.1

- fix: make sure callback functions are always used

6.28 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

6.29 version 0.12.6

- ref: DeprecationWarning: np.float from numpy 1.20

6.30 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

6.31 version 0.12.4

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

6.32 version 0.12.3

- build: migrate to GitHub Actions

6.33 version 0.12.2

- fix: properly sort curves within JPK files

6.34 version 0.12.1

- maintenance release

6.35 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)

6.36 version 0.11.0

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

6.37 version 0.10.2

- maintenance release

6.38 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*

6.39 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

6.40 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)

6.41 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

6.42 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

6.43 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

6.44 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)

6.45 version 0.5.2

- ref: drop pandas dependency (#2)

6.46 version 0.5.1

- fix: allow “force-modulation” feedback mode

6.47 version 0.5.0

- feat: meta data summary with *MetaData.get_summary*

6.48 version 0.4.1

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

6.49 version 0.4.0

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

6.50 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

6.51 version 0.2.0

- compatibility changes towards nanite

6.52 version 0.1.0

- initial version

BILBLIOGRAPHY

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

- [BAZ+22] Alice Battistella, Laura Andolfi, Michele Zanetti, Simone Dal Zilio, Marco Stebel, Giuseppe Ricci, and Marco Lazzarino. Atomic force spectroscopy-based assay to evaluate oocyte postovulatory aging. *Bioengineering & Translational Medicine*, 7(3):e10294, 2022. arXiv:<https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/btm2.10294>, doi:<https://doi.org/10.1002/btm2.10294>.
- [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, 17
- afmformats._version, 17
- afmformats.afm_data, 18
- afmformats.afm_group, 20
- afmformats.afm_qmap, 21
- afmformats.afm_segment, 23
- afmformats.errors, 24
- afmformats.formats, 25
 - afmformats.formats.fmt_chiari_txt, 25
 - afmformats.formats.fmt_hdf5, 26
 - afmformats.formats.fmt_igor, 27
 - afmformats.formats.fmt_jpk, 27
 - afmformats.formats.fmt_jpk.jpk_data, 27
 - afmformats.formats.fmt_jpk.jpk_meta, 31
 - afmformats.formats.fmt_jpk.jpk_reader, 31
 - afmformats.formats.fmt_ntmdt_txt, 33
 - afmformats.formats.fmt_tab, 33
 - afmformats.formats.fmt_workshop, 34
 - afmformats.formats.fmt_workshop.ws_map, 34
 - afmformats.formats.fmt_workshop.ws_single, 35
- afmformats.lazy_loader, 40
- afmformats.logging_setup, 40
- afmformats.meta, 41
- afmformats.mod_creep_compliance, 46
- afmformats.mod_force_distance, 47
- afmformats.mod_stress_relaxation, 48
- afmformats.parse_funcs, 49

Symbols

`__commit_id__` (in module `afmformats._version`), 18
`__version__` (in module `afmformats._version`), 17
`__version_tuple__` (in module `afmformats._version`), 17

A

`AFMCreepCompliance` (class in `afmformats.mod_creep_compliance`), 47
`AFMData` (class in `afmformats.afm_data`), 18
`AFMFileFormatError` (class in `afmformats.errors`), 24
`AFMForceDistance` (class in `afmformats.mod_force_distance`), 47
`AFMFormatRecipe` (class in `afmformats.formats`), 37
`afmformats`
 module, 17
`afmformats._version`
 module, 17
`afmformats.afm_data`
 module, 18
`afmformats.afm_group`
 module, 20
`afmformats.afm_qmap`
 module, 21
`afmformats.afm_segment`
 module, 23
`afmformats.errors`
 module, 24
`afmformats.formats`
 module, 25
`afmformats.formats.fmt_chiario_txt`
 module, 25
`afmformats.formats.fmt_hdf5`
 module, 26
`afmformats.formats.fmt_igor`
 module, 27
`afmformats.formats.fmt_jpk`
 module, 27
`afmformats.formats.fmt_jpk.jpk_data`
 module, 27
`afmformats.formats.fmt_jpk.jpk_meta`
 module, 31
`afmformats.formats.fmt_jpk.jpk_reader`
 module, 31
`afmformats.formats.fmt_ntmdt_txt`
 module, 33
`afmformats.formats.fmt_tab`
 module, 33
`afmformats.formats.fmt_workshop`
 module, 34
`afmformats.formats.fmt_workshop.ws_map`
 module, 34
`afmformats.formats.fmt_workshop.ws_single`
 module, 35
`afmformats.lazy_loader`
 module, 40
`afmformats.logging_setup`
 module, 40
`afmformats.meta`
 module, 41
`afmformats.mod_creep_compliance`
 module, 46
`afmformats.mod_force_distance`
 module, 47
`afmformats.mod_stress_relaxation`
 module, 48
`afmformats.parse_funcs`
 module, 49
`AFMGroup` (class in `afmformats.afm_group`), 21
`AFMQMap` (class in `afmformats.afm_qmap`), 22
`AFMSegment` (class in `afmformats.afm_segment`), 23
`AFMStressRelaxation` (class in `afmformats.mod_stress_relaxation`), 48
`append()` (`afmformats.afm_group.AFMGroup` method), 21
`appr` (`afmformats.mod_creep_compliance.AFMCreepCompliance` attribute), 47
`appr` (`afmformats.mod_force_distance.AFMForceDistance` attribute), 48
`appr` (`afmformats.mod_stress_relaxation.AFMStressRelaxation` attribute), 48
`ArchiveCache` (class in `afmformats.formats.fmt_jpk.jpk_reader`), 31
`as_dict()` (`afmformats.meta.MetaData` method), 42

C

`clear_cache()` (*afmformats.afm_segment.AFMSegment* method), 24

`column_dtypes` (*in module afmformats.afm_data*), 20

`column_units` (*in module afmformats.afm_data*), 20

`columns` (*afmformats.afm_data.AFMData* property), 19

`columns_innate` (*afmformats.afm_data.AFMData* property), 19

`commit_id` (*in module afmformats._version*), 18

`configure_logging()` (*in module afmformats.logging_setup*), 41

`copy()` (*afmformats.meta.MetaData* method), 42

D

`DataFileBrokenError` (*class in afmformats.errors*), 24

`DEF_ALL` (*in module afmformats.meta*), 44

`default_data_classes_by_modality` (*in module afmformats.formats*), 38

`DEFAULT_LOG_PATH` (*in module afmformats.logging_setup*), 41

`descr` (*afmformats.formats.AFMFormatRecipe* property), 37

`detect()` (*afmformats.formats.AFMFormatRecipe* method), 37

E

`enum` (*afmformats.afm_data.AFMData* property), 19

`export_data()` (*afmformats.afm_data.AFMData* method), 19

`extent` (*afmformats.afm_qmap.AFMQMap* property), 23

F

`fbool()` (*in module afmformats.parse_funcs*), 49

`feat_core_data_height_base_point_um()` (*afmformats.afm_qmap.AFMQMap* static method), 22

`feat_core_data_piezo_range_um()` (*afmformats.afm_qmap.AFMQMap* static method), 22

`feat_core_data_scan_order()` (*afmformats.afm_qmap.AFMQMap* static method), 22

`features` (*afmformats.afm_qmap.AFMQMap* attribute), 23

`FileFormatMetadataError` (*class in afmformats.errors*), 24

`FileFormatNotSupportedError` (*class in afmformats.errors*), 24

`files` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* property), 32

`find_column_dat()` (*in module afmformats.formats.fmt_jpk.jpk_data*), 27

`find_data()` (*in module afmformats.formats*), 36

`fint()` (*in module afmformats.parse_funcs*), 49

`formats_available` (*in module afmformats.formats*), 38

`formats_by_modality` (*in module afmformats.formats*), 39

`formats_by_suffix` (*in module afmformats.formats*), 38

G

`get()` (*afmformats.formats.fmt_jpk.jpk_reader.ArchiveCache* static method), 31

`get()` (*afmformats.meta.MetaData* method), 42

`get_coords()` (*afmformats.afm_qmap.AFMQMap* method), 22

`get_data()` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32

`get_enum()` (*afmformats.afm_group.AFMGroup* method), 21

`get_index_numbers()` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32

`get_index_path()` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32

`get_index_segment_numbers()` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32

`get_index_segment_path()` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32

`get_metadata()` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32

`get_modality()` (*afmformats.formats.AFMFormatRecipe* method), 37

`get_primary_meta_recipe` (*in module afmformats.formats.fmt_jpk.jpk_meta*), 31

`get_qmap()` (*afmformats.afm_qmap.AFMQMap* method), 23

`get_recipe()` (*in module afmformats.formats*), 36

`get_secondary_meta_recipe` (*in module afmformats.formats.fmt_jpk.jpk_meta*), 31

`get_summary()` (*afmformats.meta.MetaData* method), 42

`group` (*afmformats.afm_qmap.AFMQMap* attribute), 23

H

`H5DictReader` (*class in afmformats.formats.fmt_hdf5*), 26

`hierarchy` (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* property), 32

I

IMAGING_MODALITIES (in module *afmformats.meta*), 43

intr (*afmformats.mod_creep_compliance.AFMCreepCompliance* attribute), 47

intr (*afmformats.mod_stress_relaxation.AFMStressRelaxation* attribute), 48

InvalidFileFormatError (class in *afmformats.errors*), 25

items() (*afmformats.meta.MetaData* method), 43

J

JPK_COLUMNS (in module *afmformats.formats.fmt_jpk.jpk_data*), 30

JPK_SLOTS (in module *afmformats.formats.fmt_jpk.jpk_data*), 30

JPK_UNITS (in module *afmformats.formats.fmt_jpk.jpk_data*), 30

JPKReader (class in *afmformats.formats.fmt_jpk.jpk_reader*), 31

K

KEYS_VALID (in module *afmformats.meta*), 46

known_columns (in module *afmformats.afm_data*), 20

L

LazyData (class in *afmformats.lazy_loader*), 40

LazyMetaValue (class in *afmformats.meta*), 42

load_csv() (in module *afmformats.formats.fmt_workshop.ws_single*), 35

load_dat_raw() (in module *afmformats.formats.fmt_jpk.jpk_data*), 28

load_dat_unit() (in module *afmformats.formats.fmt_jpk.jpk_data*), 28

load_data() (in module *afmformats.formats*), 36

load_hdf5() (in module *afmformats.formats.fmt_hdf5*), 26

load_igor() (in module *afmformats.formats.fmt_igor*), 27

load_jpk() (in module *afmformats.formats.fmt_jpk*), 32

load_map() (in module *afmformats.formats.fmt_workshop.ws_map*), 34

load_tab() (in module *afmformats.formats.fmt_tab*), 34

load_txt() (in module *afmformats.formats.fmt_chiaro_txt*), 25

load_txt() (in module *afmformats.formats.fmt_ntmdt_txt*), 33

loader (*afmformats.formats.AFMFormatRecipe* property), 37

M

maker (*afmformats.formats.AFMFormatRecipe* property), 38

META_FIELDS (in module *afmformats.meta*), 43

meta_keys (*afmformats.errors.MissingMetaDataError* attribute), 25

metadata (*afmformats.afm_data.AFMDData* property), 19

MetaData (class in *afmformats.meta*), 42

MetaDataMissingError (class in *afmformats.meta*), 42

MissingMetaDataError (class in *afmformats.errors*), 25

modalities (*afmformats.formats.AFMFormatRecipe* property), 38

modality (*afmformats.afm_data.AFMDData* property), 19

modality (*afmformats.mod_creep_compliance.AFMCreepCompliance* property), 47

modality (*afmformats.mod_force_distance.AFMForceDistance* property), 48

modality (*afmformats.mod_stress_relaxation.AFMStressRelaxation* property), 48

module

afmformats, 17

afmformats._version, 17

afmformats.afm_data, 18

afmformats.afm_group, 20

afmformats.afm_qmap, 21

afmformats.afm_segment, 23

afmformats.errors, 24

afmformats.formats, 25

afmformats.formats.fmt_chiaro_txt, 25

afmformats.formats.fmt_hdf5, 26

afmformats.formats.fmt_igor, 27

afmformats.formats.fmt_jpk, 27

afmformats.formats.fmt_jpk.jpk_data, 27

afmformats.formats.fmt_jpk.jpk_meta, 31

afmformats.formats.fmt_jpk.jpk_reader, 31

afmformats.formats.fmt_ntmdt_txt, 33

afmformats.formats.fmt_tab, 33

afmformats.formats.fmt_workshop, 34

afmformats.formats.fmt_workshop.ws_map,

34

afmformats.formats.fmt_workshop.ws_single,

35

afmformats.lazy_loader, 40

afmformats.logging_setup, 40

afmformats.meta, 41

afmformats.mod_creep_compliance, 46

afmformats.mod_force_distance, 47

afmformats.mod_stress_relaxation, 48

afmformats.parse_funcs, 49

P

parse_time() (in module *afmformats.meta*), 41

path (*afmformats.afm_data.AFMDData* property), 20

prep (*afmformats.mod_stress_relaxation.AFMStressRelaxation* attribute), 48

Q

qmap_feature() (in module *afmformats.afm_qmap*), 22

R

ReadJPKError (class in *afmformats.formats.fmt_jpk.jpk_data*), 30
reset_data() (*afmformats.afm_data.AFMData* method), 19
retr (*afmformats.mod_creep_compliance.AFMCreepCompliance* attribute), 47
retr (*afmformats.mod_force_distance.AFMForceDistance* attribute), 48
retr (*afmformats.mod_stress_relaxation.AFMStressRelaxation* attribute), 48

S

segment (*afmformats.afm_segment.AFMSegment* attribute), 24
segment_indices (*afmformats.afm_segment.AFMSegment* property), 24
set_lazy_loader() (*afmformats.lazy_loader.LazyData* method), 40
set_metadata() (*afmformats.formats.fmt_jpk.jpk_reader.JPKReader* method), 32
shape (*afmformats.afm_qmap.AFMQMap* property), 23
subgroup_with_path() (*afmformats.afm_group.AFMGroup* method), 21
suffix (*afmformats.formats.AFMFormatRecipe* property), 38
supported_extensions (in module *afmformats.formats*), 40

U

unit_scales (in module *afmformats.afm_qmap*), 23
update() (*afmformats.meta.MetaData* method), 43

V

values() (*afmformats.meta.MetaData* method), 43
vd_str_in() (in module *afmformats.parse_funcs*), 49
version (in module *afmformats._version*), 18
version_tuple (in module *afmformats._version*), 18