FabIO Documentation

Release 0.1.3

H. Sorensen, E. Knudsen, J. Wright, G. Goret and J. Kieffer

October 27, 2013

CONTENTS

1 Getting Started **3**
- 1.1 Introduction . . . 3
- 1.2 FabIO Python module . . . 3
- 1.3 Usage . . . 5
- 1.4 Future and perspectives . . . 5
- 1.5 Conclusion . . . 5

2 Installation **9**
- 2.1 Dependencies . . . 9
- 2.2 Installation from sources . . . 9
- 2.3 Development versions . . . 10
- 2.4 Test suite . . . 10

3 Changelog **13**
- 3.1 From FabIO-0.1.2 to FabIO-0.1.3: . . . 13
- 3.2 From FabIO-0.1.1 to FabIO-0.1.2: . . . 13
- 3.3 From FabIO-0.1.0 to FabIO-0.1.1: . . . 13
- 3.4 From FabIO-0.0.8 to FabIO-0.1.0: . . . 13
- 3.5 From FabIO-0.0.7 to FabIO-0.0.8: . . . 14
- 3.6 From FabIO-0.0.6 to FabIO-0.0.7: . . . 14
- 3.7 From FabIO-0.0.4 to FabIO-0.0.6: . . . 14

4 FabIO Package **15**
- 4.1 `fabio` Package . . . 15
- 4.2 `fabio.fabioimage` Module . . . 15
- 4.3 `fabio.fabioutils` Module . . . 17
- 4.4 `fabio.file_series` Module . . . 18
- 4.5 `fabio.openimage` Module . . . 21
- 4.6 `fabio.adscimage` Module . . . 21
- 4.7 `fabio.binaryimage` Module . . . 22
- 4.8 `fabio.bruker100image` Module . . . 23
- 4.9 `fabio.brukerimage` Module . . . 23
- 4.10 `fabio.cbfimage` Module . . . 23
- 4.11 `fabio.dm3image` Module . . . 25
- 4.12 `fabio.edfimage` Module . . . 25
- 4.13 `fabio.fit2dmaskimage` Module . . . 29
- 4.14 `fabio.fit2dspreadsheetimage` Module . . . 29
- 4.15 `fabio.GEimage` Module . . . 29
- 4.16 `fabio.HiPiCimage` Module . . . 30

4.17 `fabio.kcdimage` Module 30
4.18 `fabio.mar345image` Module 30
4.19 `fabio.marccdimage` Module 31
4.20 `fabio.OXDimage` Module 31
4.21 `fabio.pilatusimage` Module 32
4.22 `fabio.pnmimage` Module 32
4.23 `fabio.tifimage` Module 32
4.24 `fabio.xsdimage` Module 33
4.25 `fabio.compression` Module 33
4.26 `fabio.converters` Module 35
4.27 `fabio.datIO` Module 36
4.28 `fabio.TiffIO` Module 36
4.29 `fabio.readbytestream` Module 36

5 Indices and tables **37**

Python Module Index **39**

Index **41**

Contents:

CHAPTER

ONE

GETTING STARTED

FabIO is a Python module for reading and handling data from two-dimensional X-ray detectors.

FabIO is a Python module written for easy and transparent reading of raw two-dimensional data from various X-ray detectors. The module provides a function for reading any image and returning a fabioimage object which contains both metadata (header information) and the raw data. All fabioimage object offer additional methods to extract information about the image and to open other detector images from the same data series.

1.1 Introduction

One obstacle when writing software to analyse data collected from a two-dimensional detector is to read the raw data into the program, not least because the data can be stored in many different formats depending on the instrument used. To overcome this problem we decided to develop a general module, FabIO (FABle I/O), to handle reading and writing of two-dimensional data. The code-base was initiated by merging parts of our fabian imageviewer and ImageD11 peak-search programs and has been developed since 2007 as part of the TotalCryst program suite for analysis of 3DXRD microscopy data. During integration into a range of scientific programs like the FABLE graphical interface, EDNA and the fast azimuthal integration library, pyFAI; FabIO has gained several features like handling multi-frame image formats as well as writing many of the file formats.

1.2 FabIO Python module

Python is a scripting language that is very popular among scientists and which also allows well structured applications and libraries to be developed.

1.2.1 Philosophy

The intention behind this development was to create a Python module which would enable easy reading of 2D data images, from any detector without having to worry about the file format. Therefore FabIO just needs a file name to open a file and it determines the file format automatically and deals with gzip and bzip2 compression transparently. Opening a file returns an object which stores the image in memory as a 2D NumPy array and the metadata, called header, in a Python dictionary. Beside the data and header attributes, some methods are provided for reading the previous or next image in a series of images as well as jumping to a specific file number. For the user, these auxiliary methods are intended to be independent of the image format (as far as is reasonably possible).

FabIO is written in an object-oriented style (with classes) but aims at being used in a scripting environment: special care has been taken to ensure the library remains easy to use. Therefore no knowledge of object-oriented programming is required to get full benefits of the library. As the development is done in a collaborative and decentralized way; a comprehensive test suite has been added to reduce the number of regressions when new features are added or old problems are repaired. The software is very modular and allows new classes to be added for handling other data formats easily. FabIO and its source-code are freely available to everyone on-line, licensed under the GNU General Public License version 3 (GPLv3). FabIO is also available directly from popular Linux distributions like Debian and Ubuntu.

1.2.2 Implementation

The main language used in the development of FabIO is Python; however, some image formats are compressed and require compression algorithms for reading and writing data. When such algorithms could not be implemented efficiently using Python or NumPy native modules were developed, in i.e. standard C code callable from Python (sometimes generated using Cython). This code has to be compiled for each computer architecture and offers excellent performance. FabIO is only dependent on the NumPy module and has extra features if two other optional Python modules are available. For reading XML files (that are used in EDNA) the Lxml module is required and the Python Image Library, PIL is needed for producing a PIL image for displaying the image in graphical user interfaces and several image-processing operations that are not re-implemented in FabIO. A variety of useful image processing is also available in the scipy.ndimage module and in scikits-image.

Images can also be displayed in a convenient interactive manner using matplotlib and an IPython shell , which is mainly used for developing data analysis algorithms. Reading and writing procedure of the various TIFF formats is based on the TiffIO code from PyMCA.

In the Python shell, the *fabio* module must be imported prior to reading an image in one of the supported file formats (see Table *Supported formats*, hereafter). The *fabio.open* function creates an instance of the Python class *fabioimage*, from the name of a file. This instance, named *img* hereafter, stores the image data in *img.data* as a 2D NumPy array. Often the image file contains more information than just the intensities of the pixels, e.g. information about how the image is stored and the instrument parameters at the time of the image acquisition, these metadata are usually stored in the file header. Header information, are available in *img.header* as a Python dictionary where keys are strings and values are usually strings or numeric values.

Information in the header about the binary part of the image (compression, endianness, shape) are interpreted however, other metadata are exposed as they are recorded in the file. FabIO allows the user to modify and, where possible, to save this information (the table *Supported formats* summarizes writable formats). Automatic translation between file-formats, even if desirable, is sometimes impossible because not all format have the capability to be extended with additional metadata. Nevertheless FabIO is capable of converting one image data-format into another by taking care of the numerical specifics: for example float arrays are converted to integer arrays if the output format only accepts integers.

1.2.3 FabIO methods

One strength of the implementation in an object oriented language is the possibility to combine functions (or methods) together with data appropriate for specific formats. In addition to the header information and image data, every *fabioimage* instance (returned by *fabio.open*) has methods inherited from *fabioimage* which provide information about the image minimum, maximum and mean values. In addition there are methods which return the file number, name etc. Some of the most important methods are specific for certain formats because the methods are related to how frames in a sequence are handled; these methods are *img.next()*, *img.previous()*, and *img.getframe(n)*. The behaviour of such methods varies depending on the image format: for single-frame format (like mar345), *img.next()* will return the image in next file; for multi-frame format (like GE), *img.next()* will return the next frame within the same file. For formats which are possibly multi-framed like EDF, the behaviour depends on the actual number of frames per file (accessible via the *img.nframes* attribute).

1.3 Usage

1.3.1 Examples

In this section we have collected some basic examples of how FabIO can be employed.

Opening an image:

```
import fabio
im100 = fabio.open('Quartz_0100.tif') # Open image file
print(im0.data[1024,1024])            # Check a pixel value
im101 = im100.next()                  # Open next image
im270 = im1.getframe(270)             # Jump to file number 270: Quartz_0270.tif
```

Normalising the intensity to a value in the header:

```
img = fabio.open('exampleimage0001.edf')
print(img.header)
{'ByteOrder': 'LowByteFirst',
 'DATE (scan begin)': 'Mon Jun 28 21:22:16 2010',
 'ESRFCurrent': '198.099',
...
}
# Normalise to beam current and save data
srcur = float(img.header['ESRFCurrent'])
img.data *= 200.0/srcur
img.write('normed_0001.edf')
```

Interactive viewing with matplotlib:

```
from matplotlib import pyplot       # Load matplotlib
pyplot.imshow(img.data)             # Display as an image
pyplot.show()                       # Show GUI window
```

1.4 Future and perspectives

The Hierarchical Data Format version 5 (*hdf5*) is a data format which is increasingly popular for storage of X-ray and neutron data. To name a few facilities the synchrotron Soleil and the neutron sources ISIS, SNS and SINQ already use HDF extensively through the NeXus format. For now, mainly processed or curated data are stored in this format but new detectors are rumoured to provide native output in HDF5. FabIO will rely on H5Py, which already provides a good HDF5 binding for Python, as an external dependency, to be able to read and write such HDF5 files.

In the near future FabIO will be upgraded to work with Python3 (a new version of Python); this change of version will affect some internals FabIO as string and file handling have been altered. This change is already ongoing as many parts of native code in C have already been translated into Cython to smoothe the transition, since Cython generates code compatible with Python3. This also makes it easier to retain backwards compatibility with the earlier Python versions.

1.5 Conclusion

FabIO gives an easy way to read and write 2D images when using the Python computer language. It was originally developed for X-ray diffraction data but now gives an easy way for scientists to access and manipulate their data from a wide range of 2D X-ray detectors. We welcome contributions to further improve the code and hope to add more file formats in the future as well as port the existing code base to the emerging Python3.

1.5.1 Acknoledgements

We acknowledge Andy Götz and Kenneth Evans for extensive testing when including the FabIO reader in the Fable image viewer (Götz et al., 2007).We also thank V. Armando Solé for assistance with his TiffIO reader and Carsten Gundlach for deployment of FabIO at the beamlines i711 and i811, MAX IV, and providing bug reports. We finally acknowledge our colleagues who have reported bugs and helped to improve FabIO. Financial support was granted by the EU 6th Framework NEST/ADVENTURE project TotalCryst (Poulsen et al., 2006).

1.5.2 Citation

Knudsen, E. B., Sørensen, H. O., Wright, J. P., Goret, G. & Kieffer, J. (2013). J. Appl. Cryst. 46, 537-539.

http://dx.doi.org/10.1107/S0021889813000150

1.5.3 List of file formats that FabIO can read and write

In alphabetical order. The listed filename extensions are typical examples. FabIO tries to deduce the actual format from the file itself and only uses extensions as a fallback if that fails.

Table 1.1: Supported formats

Python Module	Detector / Format	Extension	Read	Multi-image	Write
ADSC	ADSC Quantum	.img	Yes	No	Yes
Bruker	Bruker formats	.sfrm	Yes	No	Yes
DM3	Gatan Digital Micrograph	.dm3	Yes	No	No
EDF	ESRF data format	.edf	Yes	Yes	Yes
EDNA-XML	Used by EDNA	.xml	Yes	No	No
CBF	CIF binary files	.cbf	Yes	No	Yes
kcd	Nonius KappaCCD	.kccd	Yes	No	No
fit2d mask	Used by Fit2D	.msk	Yes	No	Yes
fit2d spreadsheet	Used by Fit2D	.spr	Yes	No	Yes
GE	General Electric	No	Yes	Yes	No
HiPiC	Hamamatsu CCD	.tif	Yes	No	No
marccd	MarCCD/Mar165	.mccd	Yes	No	Yes
mar345	Mar345 image plate	.mar3450	Yes	No	Yes
OXD	Oxford Diffraction	.img	Yes	No	Yes
pilatus	Dectris Pilatus Tiff	.tif	Yes	No	Yes
PNM	Portable aNy Map	.pnm	Yes	No	No
TIFF	Tagged Image File Format	.tif	Yes	No	Yes

1.5.4 Adding new file formats

We hope it will be relatively easy to add new file formats to fabio in the future. The basic idea is the following:

1. inherit from fabioimage overriding the methods _readheader, read and optionally write. Name your new module XXXimage where XXX means something (eg tifimage).
2. readheader fills in a dictionary of "name":"value" pairs in self.header. No one expects to find anything much in there.

3. read fills in self.data with a numpy array holding the image. Some redundant info which also appears are self.dim1 and self.dim2: the image dimensions, self.bpp is the bytes per pixel and self.bytecode is the numpy.dtype.type of the data.
4. The member variables "_need_a_seek_to_read" and "_need_a_real_file" are there in case you have trouble with the transparent handling of bz2 and gz files.
5. Register the file type (extension naming) in fabioutils.py:FILETYPES
6. Add your new module as an import into fabio.openimage
7. Fill out the magic numbers for your format in fabio.openimage if you know them (the characteristic first few bytes in the file)
8. Upload a testimage to the file release system and create a unittest testcase which opens an example of your new format, confirming the image has actually been read in successfully (eg check the mean, max, min and esd are all correct, perhaps orientation too)
9. Run pylint on your code and then please go clean it up. Have a go at mine while you are at it.
10. Bask in the warm glow of appreciation when someone unexpectedly learns they don't need to convert their data into another format

CHAPTER

TWO

INSTALLATION

FabIO can, as any Python module, be installed from its sources, available on sourceforge but we advice to use binary packages provided for the most common platforms on sourceforge: Windows, MacOSX and Linux. Moreover FabIO is part of the common Linux distributions Ubuntu (since 11.10) and Debian7 where the package is named python-fabio and can be installed via:

```
# apt-get install python-fabio
```

If you are using MS Windows or MacOSX; binary version have been packaged. Windows installers are executable, just download the one corresponding to you python version and run it. MacOSX builds are zipped: unzip them at the right place.

2.1 Dependencies

- Python 2.5 or later (python 3.x is not yet ready)
- numpy - http://www.numpy.org

For full functionality of Fabio the following modules need to be installed:

- PIL (python imaging library) - http://www.pythonware.com
- lxml (library for reading XSDimages)

2.2 Installation from sources

FabIO can be downloaded from the fable download page on sourceforge.net. Presently the source code has been distributed as a zip package and a compressed tarball. Download either one and unpack it.

```
http://sourceforge.net/projects/fable/files/fabio/
```

e.g.

```
tar xvzf fabio-0.1.3.tar.gz
```

or

```
unzip fabio-0.1.3.zip
```

all files are unpacked into the directory fabio-0.1.3. To install these do

```
cd fabio-0.1.3
```

and install fabio with

```
python setup.py build
sudo python setup.py install
```

most likely you will need to gain root privileges (with sudo in front of the command) to install the built package.

2.3 Development versions

The newest development version can be obtained by checking it out from the subversion (SVN) repository:

```
svn checkout https://svn.sourceforge.net/svnroot/fable/fabio/trunk fabio
cd fabio
python setup.py build
sudo python setup.py install
```

For Ubuntu/Debian users, you will need:

- python-imaging
- python-imaging-tk
- python-numpy
- python-dev

```
sudo apt-get install python-imaging python-imaging-tk python-numpy
```

We provide also a debian-package builder based on stdeb:

```
sudo apt-get install python-stdeb
./build-deb.sh
```

which builds a debian package and installs it in a single command. Handy for testing.

2.4 Test suite

FabIO has a comprehensive test-suite to ensure non regression (about 100 tests). When you run the test for the first time, many test images will be download and converted into various compressed format like gzip and bzip2 (this takes a lot of time). Be sure you have an internet connection (and your environment variable http_proxy is correctly set-up, if you are behind a proxy).

```
python setup.py build
cd test
python test_all.py
........................................WARNING:compression:Encounter the python-gzip bug with traili
..............................WARNING:edfimage:Non complete datablock: got 6928, expected 8388608
WARNING:edfimage:Non complete datablock: got 6928, expected 8388608
WARNING:edfimage:Non complete datablock: got 6928, expected 8388608
.....................WARNING:edfimage:Unknown compression scheme TY1
.....WARNING:edfimage:Unknown compression scheme FALSE
...WARNING: Non standard TIFF. Rows per strip TAG missing
WARNING: Non standard TIFF. Strip byte counts TAG missing
....
----------------------------------------------------------------------
```

```
Ran 103 tests in 21.696s
OK
```

Many tests are there to deal with malformed files, don’t worry if the programs comaplins in warnings about “bad files”, it is done on purpose.

CHAPTER

THREE

CHANGELOG

3.1 From FabIO-0.1.2 to FabIO-0.1.3:

- Fixed a memory-leak in mar345 module
- Improved support for bruker format (writer & reader)
- Fixed a bug in EDF headers (very long headers)
- Provide template for new file-formats
- Fix a bug related to PIL in new MacOSX
- Allow binary-images to be read from end

3.2 From FabIO-0.1.1 to FabIO-0.1.2:

- Fixed a bug in fabioimage.write (impacted all writers)
- added Sphinx documentation “python setup.py build_doc”
- PyLint compliance of some classes (rename, ...)
- tests from installer with “python setup.py build test”

3.3 From FabIO-0.1.0 to FabIO-0.1.1:

- Merged Mar345 image reader and writer with cython bindings (towards python3 compliance)
- Improve CBF image writing under windows
- Bz2, Gzip and Flat files are managed through a common way ... classes are more (python v2.5) or less (python v2.7) overloaded
- Fast EDF reading if one assumes offsets are the same between files, same for ROIs

3.4 From FabIO-0.0.8 to FabIO-0.1.0:

- OXD reader improved and writer implemented
- Mar345 reader improved and writer implemented

- CBF writer implemented
- Clean-up of the code & bug fixes
- Move towards python3
- Make PIL optional dependency

Python3 is not yet tested but some blocking points have been identified and some fixed.

3.5 From FabIO-0.0.7 to FabIO-0.0.8:

- Support for Tiff using TiffIO module from V.A.Solé
- Clean-up of the code & bug fixes

3.6 From FabIO-0.0.6 to FabIO-0.0.7:

- Support for multi-frames EDF files
- Support for XML images/2D arrays used in EDNA
- new method: fabio.open(filename) that is an alias for fabio.openimage.openimage(filename)

3.7 From FabIO-0.0.4 to FabIO-0.0.6:

- Support for CBF files from Pilatus detectors
- Support for KCD files from Nonius Kappa CCD images
- write EDF with their native data type (instead of uint16 by default)

CHAPTER

FOUR

FABIO PACKAGE

4.1 `fabio` Package

FabIO module

4.2 `fabio.fabioimage` Module

Authors: Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk

and Jon Wright, Jerome Kieffer: ESRF

class `fabio.fabioimage.`**`fabioimage`**(*data=None*, *header=None*)

Bases: `object`

A common object for images in fable Contains a numpy array (.data) and dict of meta data (.header)

`add`(*other*)

Add another Image - warning, does not clip to 16 bit images by default

static `checkData`(*data=None*)

Empty for fabioimage but may be populated by others classes, especially for format accepting only integers

static `checkHeader`(*header=None*)

Empty for fabioimage but may be populated by others classes

`classname`

Retrieves the name of the class :return: the name of the class

`convert`(*dest*)

Convert a fabioimage object into another fabioimage object (with possible conversions) :param dest: destination type “EDF”, “edfimage” or the class itself

`getclassname`()

Retrieves the name of the class :return: the name of the class

`getframe`(*num*)

returns the file numbered ‘num’ in the series as a fabioimage

`getheader`()

returns self.header

`getmax`()

Find max value in self.data, caching for the future

getmean()
return the mean

getmin()
Find min value in self.data, caching for the future

getstddev()
return the standard deviation

integrate_area(*coords*)
Sums up a region of interest if len(coords) == 4 -> convert coords to slices if len(coords) == 2 -> use as slices floor -> ? removed as unused in the function.

load(**arg*, ***kwarg*)
Wrapper for read

make_slice(*coords*)
Convert a len(4) set of coords into a len(2) tuple (pair) of slice objects the latter are immutable, meaning the roi can be cached

next()
returns the next file in the series as a fabioimage

previous()
returns the previous file in the series as a fabioimage

read(*filename*, *frame=None*)
To be overridden - fill in self.header and self.data

readROI(*filename*, *frame=None*, *coords=None*)
Method reading Region of Interest. This implementation is the trivial one, just doing read and crop

readheader(*filename*)
Call the _readheader function...

rebin(*x_rebin_fact*, *y_rebin_fact*, *keep_I=True*)
Rebin the data and adjust dims :param x_rebin_fact: x binning factor :param y_rebin_fact: y binning factor :param keep_I: shall the signal increase ? :type x_rebin_fact: int :type y_rebin_fact: int :type keep_I: boolean

resetvals()
Reset cache - call on changing data

save(*fname*)
wrapper for write

toPIL16(*filename=None*)
Convert to Python Imaging Library 16 bit greyscale image

FIXME - this should be handled by the libraries now

update_header(***kwds*)
update the header entries by default pass in a dict of key, values.

write(*fname*)
To be overwritten - write the file

fabio.fabioimage.**test**()
check some basic fabioimage functionality

4.3 `fabio.fabioutils` Module

General purpose utilities functions for fabio

class `fabio.fabioutils.`**`BZ2File`**(*name*, *mode='r'*, *buffering=0*, *compresslevel=9*)

Bases: `bz2.BZ2File`

Wrapper with lock

`getSize`()

`setSize`(*value*)

`size`

class `fabio.fabioutils.`**`File`**(*name*, *mode='rb'*, *buffering=0*)

Bases: `file`

wrapper for “file” with locking

`getSize`()

`setSize`(*size*)

`size`

class `fabio.fabioutils.`**`FilenameObject`**(*stem=None*, *num=None*, *directory=None*, *format=None*, *extension=None*, *postnum=None*, *digits=4*, *filename=None*)

Bases: `object`

The ‘meaning’ of a filename ...

`deconstruct_filename`(*filename*)

Break up a filename to get image type and number

`str`()

Return a string representation

`tostring`()

convert yourself to a string

class `fabio.fabioutils.`**`GzipFile`**(*filename=None*, *mode=None*, *compresslevel=9*, *fileobj=None*)

Bases: `gzip.GzipFile`

Just a wrapper forgzip.GzipFile providing the correct seek capabilities for python 2.5

class `fabio.fabioutils.`**`StringIO`**(*data*, *fname=None*, *mode='r'*)

Bases: `StringIO.StringIO`

just an interface providing the name and mode property to a StringIO

BugFix for MacOSX mainly

`getSize`()

`setSize`(*size*)

`size`

class `fabio.fabioutils.`**`UnknownCompressedFile`**(*name*, *mode='rb'*, *buffering=0*)

Bases: `fabio.fabioutils.File`

wrapper for “File” with locking

`fabio.fabioutils.`**`construct_filename`**(*filename*, *frame=None*)

Try to construct the filename for a given frame

`fabio.fabioutils.`**`deconstruct_filename`**(*filename*)
Function for backward compatibility. Deprecated

`fabio.fabioutils.`**`deprecated`**(*func*)
used to deprecate a function/method: prints a lot of warning messages to enforce the modifaction of the code

`fabio.fabioutils.`**`extract_filenumber`**(*name*)
extract file number

`fabio.fabioutils.`**`getnum`**(*name*)
# try to figure out a file number # guess it starts at the back

`fabio.fabioutils.`**`isAscii`**(*name*, *listExcluded=None*)

Parameters

- **name** – string to check
- **listExcluded** – list of char or string excluded.

Returns True of False whether name is pure ascii or not

`fabio.fabioutils.`**`jump_filename`**(*name*, *num*, *padding=True*)
jump to number

`fabio.fabioutils.`**`next_filename`**(*name*, *padding=True*)
increment number

`fabio.fabioutils.`**`nice_int`**(*s*)
Workaround that int('1.0') raises an exception

Parameters **s** – string to be converted to integer

`fabio.fabioutils.`**`numstem`**(*name*)
cant see how to do without reversing strings Match 1 or more digits going backwards from the end of the string

`fabio.fabioutils.`**`pad`**(*mystr*, *pattern=' '*, *size=80*)
Performs the padding of the string to the right size with the right pattern

`fabio.fabioutils.`**`previous_filename`**(*name*, *padding=True*)
decrement number

`fabio.fabioutils.`**`toAscii`**(*name*, *excluded=None*)

Parameters

- **name** – string to check
- **excluded** – tuple of char or string excluded (not list: they are mutable).

Returns the name with all non valid char removed

4.4 `fabio.file_series` Module

4.4.1 Authors:

- Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk
- Jon Wright, ESRF

class `fabio.file_series.`**`file_series`**(*list_of_strings*)

Bases: `list`

Represents a series of files to iterate has an idea of a current position to do next and prev

You also get from the list python superclass: append count extend insert pop remove reverse sort

`current`()

Current position in a sequence

`current_image`()

Current image in sequence

Returns fabioimage

`current_object`()

Current image in sequence

Returns file_object

`first`()

First image in series

`first_image`()

First image in a sequence

Returns fabioimage

`first_object`()

First image in a sequence

Returns file_object

`jump`(*num*)

Goto a position in sequence

`jump_image`(*num*)

Jump to and read image

Returns fabioimage

`jump_object`(*num*)

Jump to and read image

Returns file_object

`last`()

Last in series

`last_image`()

Last image in a sequence

Returns fabioimage

`last_object`()

Last image in a sequence

Returns file_object

`len`()

Number of files

`next`()

Next in a sequence

next_image()
Return the next image

Returns fabioimage

next_object()
Return the next image

Returns file_object

previous()
Prev in a sequence

previous_image()
Return the previous image

Returns fabioimage

previous_object()
Return the previous image

Returns file_object

class `fabio.file_series.`**filename_series**(*filename*)
Much like the others, but created from a string filename

current()
return current filename string

current_image()
returns the current image as a fabioimage

current_object()
returns the current filename as a fabio.FilenameObject

jump(*num*)
jump to a specific number

jump_image(*num*)
returns the image number as a fabioimage

jump_object(*num*)
returns the filename num as a fabio.FilenameObject

next()
increment number

next_image()
returns the next image as a fabioimage

next_object()
returns the next filename as a fabio.FilenameObject

prev_image()
returns the previos image as a fabioimage

previous()
decrement number

previous_object()
returns the previous filename as a fabio.FilenameObject

`fabio.file_series.`**new_file_series**(*first_object*, *nimages=0*, *step=1*, *traceback=False*)
A generator function that creates a file series starting from a a fabioimage. Iterates through all images in a file (if more than 1), then proceeds to the next file as determined by fabio.next_filename.

Parameters

- **first_object** – the starting fabioimage, which will be the first one yielded in the sequence
- **nimages** – the maximum number of images to consider step: step size, will yield the first and every step'th image until nimages is reached. (e.g. nimages = 5, step = 2 will yield 3 images (0, 2, 4)
- **traceback** – if True causes it to print a traceback in the event of an exception (missing image, etc.). Otherwise the calling routine can handle the exception as it chooses
- **yields** – the next fabioimage in the series. In the event there is an exception, it yields the sys.exec_info for the exception instead. sys.exec_info is a tuple: (exceptionType, exceptionValue, exceptionTraceback) from which all the exception information can be obtained.

Suggested usage:

```
for obj in new_file_series( ... ):
  if not isinstance(obj, fabio.fabioimage.fabioimage ):
    # deal with errors like missing images, non readable files, etc
    # e.g.
    traceback.print_exception(obj[0], obj[1], obj[2])
```

`fabio.file_series.`**`new_file_series0`**(*first_object*, *first=None*, *last=None*, *step=1*)
Created from a fabio image first and last are file numbers

class `fabio.file_series.`**`numbered_file_series`**(*stem*, *first*, *last*, *extension*, *digits=4*, *padding='Y'*, *step=1*)
Bases: `fabio.file_series.file_series`

mydata0001.edf = "mydata" + 0001 + ".edf" mydata0002.edf = "mydata" + 0002 + ".edf" mydata0003.edf = "mydata" + 0003 + ".edf"

4.5 `fabio.openimage` Module

Authors: Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:henning.sorensen@risoe.dk

mods for fabio by JPW

`fabio.openimage.`**`do_magic`**(*byts*)
Try to interpret the bytes starting the file as a magic number

`fabio.openimage.`**`openheader`**(*filename*)
return only the header

`fabio.openimage.`**`openimage`**(*filename*, *frame=None*)
Try to open an image

4.6 `fabio.adscimage` Module

Authors: Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk

- mods for fabio by JPW

class `fabio.adscimage.`**`adscimage`**(**args*, ***kwargs*)

Bases: `fabio.fabioimage.fabioimage`

Read an image in ADSC format (quite similar to edf?)

`read`(*fname*, *frame=None*)

read in the file

`write`(*fname*)

Write adsc format

`fabio.adscimage.`**`test`**()

testcase

4.7 `fabio.binaryimage` Module

Authors: Gael Goret, Jerome Kieffer, ESRF, France Emails: gael.goret@esrf.fr, jerome.kieffer@esrf.fr

Brian Richard Pauw <brian@stack.nl>

Binary files images are simple none-compressed 2D images only defined by their : data-type, dimensions, byte order and offset

This simple library has been made for manipulating exotic/unknown files format.

class `fabio.binaryimage.`**`binaryimage`**(**args*, ***kwargs*)

Bases: `fabio.fabioimage.fabioimage`

This simple library has been made for manipulating exotic/unknown files format.

Binary files images are simple none-compressed 2D images only defined by their : data-type, dimensions, byte order and offset

if offset is set to a negative value, the image is read using the last data but n data in the file, skipping any header.

`estimate_offset_value`(*fname*, *dim1*, *dim2*, *bytecode='int32'*)

Estimates the size of a file

`read`(*fname*, *dim1*, *dim2*, *offset=0*, *bytecode='int32'*, *endian='<'*)

Read a binary image

Parameters

- **fname** (*string*) – file name
- **dim1** – image dimensions (Fast index)
- **dim2** – image dimensions (Slow index)
- **offset** – starting position of the data-block. If negative, starts at the end.
- **bytecode** – can be “int8”,”int16”,”int32”,”int64”,”uint8”,”uint16”,”uint32”,”uint64”,”float32”,”float64”,...
- **endian** – among short or long endian (“<” or “>”)

static **`swap_needed`**(*endian*)

Decide if we need to byteswap

`write`(*fname*)

4.8 `fabio.bruker100image` Module

class `fabio.bruker100image.`**`bruker100image`**(*data=None*, *header=None*)

Bases: `fabio.brukerimage.brukerimage`

`read`(*fname*, *frame=None*)

`toPIL16`(*filename=None*)

4.9 `fabio.brukerimage` Module

Authors: Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk

Based on: openbruker,readbruker, readbrukerheader functions in the opendata module of ImageD11 written by Jon Wright, ESRF, Grenoble, France

Writer by Jérôme Kieffer, ESRF, Grenoble, France

class `fabio.brukerimage.`**`brukerimage`**(*data=None*, *header=None*)

Bases: `fabio.fabioimage.fabioimage`

Read and eventually write ID11 bruker (eg smart6500) images

TODO: int32 -> float32 conversion according to the "linear" keyword. This is done and works but we need to check with other program that we are appliing the right formula and not the reciprocal one.

`HEADERS_KEYS` **= ['FORMAT', 'VERSION', 'HDRBLKS', 'TYPE', 'SITE', 'MODEL', 'USER', 'SAMPLE', 'SETNAM**

`SPACER` **= '\x1a\x04'**

`basic_translate`(*fname=None*)

Does some basic population of the headers so that the writing is possible

`bpp_to_numpy` **= {1: <type 'numpy.uint8'>, 2: <type 'numpy.uint16'>, 4: <type 'numpy.uint32'>}**

`calc_bpp`(*data=None*, *max_entry=4096*)

Calculate the number of byte per pixel to get an optimal overflow table.

Returns byte per pixel

`gen_header`()

Generate headers (with some magic and guesses) :param format can be 86 or 100

`gen_overflow`()

Generate an overflow table

`read`(*fname*, *frame=None*)

Read in and unpack the pixels (including overflow table

`write`(*fname*)

Write a bruker image

`fabio.brukerimage.`**`test`**()

a testcase

4.10 `fabio.cbfimage` Module

Authors: Jérôme Kieffer, ESRF email:jerome.kieffer@esrf.fr

Cif Binary Files images are 2D images written by the Pilatus detector and others. They use a modified (simplified) byte-offset algorithm.

CIF is a library for manipulating Crystallographic information files and tries to conform to the specification of the IUCR

class `fabio.cbfimage.`**`CIF`**(*_strFilename=None*)

Bases: `dict`

This is the CIF class, it represents the CIF dictionary; and as a a python dictionary thus inherits from the dict built in class.

`BINARY_MARKER` = ‘–CIF-BINARY-FORMAT-SECTION–‘

`BLANK` = [’ ‘, ‘\t’, ‘\r’, ‘\n’, ‘\r\n’, ‘\n\r’]

`EOL` = [’\r’, ‘\n’, ‘\r\n’, ‘\n\r’]

static `LoopHasKey`(*loop*, *key*)

Returns True if the key (string) exist in the array called loop

`START_COMMENT` = [’”’, ‘“”’]

`exists`(*sKey*)

Check if the key exists in the CIF and is non empty. :param sKey: CIF key :type sKey: string :param cif: CIF dictionary :return: True if the key exists in the CIF dictionary and is non empty :rtype: boolean

`existsInLoop`(*sKey*)

Check if the key exists in the CIF dictionary. :param sKey: CIF key :type sKey: string :param cif: CIF dictionary :return: True if the key exists in the CIF dictionary and is non empty :rtype: boolean

static `isAscii`(*_strIn*)

Check if all characters in a string are ascii,

Parameters **_strIn** (*python string*) – input string

Returns boolean

Return type boolean

`loadCHIPLOT`(*_strFilename*)

Load the powder diffraction CHIPLOT file and returns the pd_CIF dictionary in the object

Parameters **_strFilename** (*string*) – the name of the file to open

Returns the CIF object corresponding to the powder diffraction

Return type dictionary

`loadCIF`(*_strFilename*, *_bKeepComment=False*)

Load the CIF file and populates the CIF dictionary into the object :param _strFilename: the name of the file to open :type _strFilename: string :param _strFilename: the name of the file to open :type _strFilename: string :return: None

`pop`(*key*)

`popitem`(*key*)

`readCIF`(*_strFilename*, *_bKeepComment=False*)

Load the CIF file and populates the CIF dictionary into the object :param _strFilename: the name of the file to open :type _strFilename: string :param _strFilename: the name of the file to open :type _strFilename: string :return: None

`saveCIF`(*_strFilename=’test.cif’*, *linesep=’n’*, *binary=False*)

Transforms the CIF object in string then write it into the given file :param _strFilename: the of the file to be written :param linesep: line separation used (to force compatibility with windows/unix) :param binary: Shall we write the data as binary (True only for imageCIF/CBF) :type param: string

`tostring``(`*_strFilename=None*, *linesep='n'*`)`

Converts a cif dictionnary to a string according to the CIF syntax

Parameters **_strFilename** (*string*) – the name of the filename to be appended in the header of the CIF file

Returns a sting that corresponds to the content of the CIF - file.

class `fabio.cbfimage.`**`cbfimage`**`(`*data=None*, *header=None*, *fname=None*`)`

Bases: `fabio.fabioimage.fabioimage`

Read the Cif Binary File data format

static **`checkData`**`(`*data=None*`)`

`read``(`*fname*, *frame=None*`)`

Read in header into self.header and the data into self.data

`write``(`*fname*`)`

write the file in CBF format :param fname: name of the file :type: string

4.11 `fabio.dm3image` Module

Authors: Henning O. Sorensen & Erik Knudsen

Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk

- Jon Wright, ESRF

class `fabio.dm3image.`**`dm3image`**`(`**args*, ***kwargs*`)`

Bases: `fabio.fabioimage.fabioimage`

Read and try to write the dm3 data format

`read``(`*fname*, *frame=None*`)`

`read_data``()`

`read_tag_entry``()`

`read_tag_group``()`

`read_tag_type``()`

`readbytes``(`*bytes_to_read*, *format*, *swap=True*`)`

4.12 `fabio.edfimage` Module

License: GPLv2+

4.12.1 Authors:

- Henning O. Sorensen & Erik Knudsen: Center for Fundamental Research: Metal Structures in Four Dimensions; Risoe National Laboratory; Frederiksborgvej 399; DK-4000 Roskilde; email:erik.knudsen@risoe.dk

- Jon Wright & Jérôme Kieffer: European Synchrotron Radiation Facility; Grenoble (France)

class `fabio.edfimage.`**`Frame`**(*data=None*, *header=None*, *header_keys=None*, *number=None*)

Bases: `object`

A class representing a single frame in an EDF file

`bytecode`

`data`

Unpack a binary blob according to the specification given in the header

Returns dataset as numpy.ndarray

`getByteCode`()

`getData`()

Unpack a binary blob according to the specification given in the header

Returns dataset as numpy.ndarray

`getEdfBlock`(*force_type=None*, *fit2dMode=False*)

Parameters

- **force_type** (*string or numpy.dtype*) – type of the dataset to be enforced like “float64” or “uint16”
- **fit2dMode** (*boolean*) – enforce compatibility with fit2d and starts countimg number of images at 1

Returns ascii header block

Return type python string with the concatenation of the ascii header and the binary data block

`parseheader`(*block*)

Parse the header in some EDF format from an already open file

Parameters **block** (*string, should be full ascii*) – string representing the header block

Returns size of the binary blob

`setByteCode`(*_iVal*)

`setData`(*npa=None*)

Setter for data in edf frame

`swap_needed`()

Decide if we need to byteswap

class `fabio.edfimage.`**`edfimage`**(*data=None*, *header=None*, *header_keys=None*, *frames=None*)

Bases: `fabio.fabioimage.fabioimage`

Read and try to write the ESRF edf data format

`appendFrame`(*frame=None*, *data=None*, *header=None*)

Method used add a frame to an EDF file :param frame: frame to append to edf image :type frame: instance of Frame :return: None

`bpp`

`bytecode`

`capsHeader`

property: capsHeader of EDF file, i.e. the keys of the header in UPPER case.

static `checkHeader`(*header=None*)
Empty for fabioimage but may be populated by others classes

`data`
property: data of EDF file

`delCapsHeader`()
deleter for edf capsHeader

`delData`()
deleter for edf Data

`delHeader`()
Deleter for edf header

`delHeaderKeys`()
Deleter for edf header_keys

`deleteFrame`(*frameNb=None*)
Method used to remove a frame from an EDF image. by default the last one is removed. :param frameNb: frame number to remove, by default the last. :type frameNb: integer :return: None

`dim1`

`dim2`

`dims`

`fastReadData`(*filename=None*)
This is a special method that will read and return the data from another file ... The aim is performances, ... but only supports uncompressed files.

Returns data from another file using positions from current edfimage

`fastReadROI`(*filename*, *coords=None*)
Method reading Region of Interest of another file based on metadata available in current edfimage. The aim is performances, ... but only supports uncompressed files.

Returns ROI-data from another file using positions from current edfimage

Return type numpy 2darray

`getBpp`()

`getByteCode`()

`getCapsHeader`()
getter for edf headers keys in upper case :return: data for current frame :rtype: dict

`getData`()
getter for edf Data :return: data for current frame :rtype: numpy.ndarray

`getDim1`()

`getDim2`()

`getDims`()

`getHeader`()
Getter for the headers. used by the property header,

`getHeaderKeys`()
Getter for edf header_keys

`getNbFrames`()
Getter for number of frames

getframe(*num*)
returns the file numbered 'num' in the series as a fabioimage

header
property: header of EDF file

header_keys
property: header_keys of EDF file

next()
returns the next file in the series as a fabioimage

nframes
Getter for number of frames

previous()
returns the previous file in the series as a fabioimage

read(*fname*, *frame=None*)
Read in header into self.header and the data into self.data

setBpp(*_iVal*)

setByteCode(*_iVal*)

setCapsHeader(*_data*)
Enforces the propagation of the header_keys to the list of frames :param _data: numpy array representing data

setData(*_data*)
Enforces the propagation of the data to the list of frames :param _data: numpy array representing data

setDim1(*_iVal*)

setDim2(*_iVal*)

setHeader(*_dictHeader*)
Enforces the propagation of the header to the list of frames

setHeaderKeys(*_listtHeader*)
Enforces the propagation of the header_keys to the list of frames :param _listtHeader: list of the (ordered) keys in the header :type _listtHeader: python list

setNbFrames(*val*)
Setter for number of frames ... should do nothing. Here just to avoid bugs

swap_needed()
Decide if we need to byteswap

unpack()
Unpack a binary blob according to the specification given in the header and return the dataset

Returns dataset as numpy.ndarray

write(*fname*, *force_type=None*, *fit2dMode=False*)
Try to write a file check we can write zipped also mimics that fabian was writing uint16 (we sometimes want floats)

Parameters **force_type** – can be numpy.uint16 or simply "float"

Returns None

4.13 `fabio.fit2dmaskimage` Module

Author: Andy Hammersley, ESRF Translation into python/fabio: Jon Wright, ESRF

class `fabio.fit2dmaskimage.`**`fit2dmaskimage`**(*data=None*, *header=None*)

Bases: `fabio.fabioimage.fabioimage`

Read and try to write Andy Hammersley's mask format

static **`checkData`**(*data=None*)

`read`(*fname*, *frame=None*)

Read in header into self.header and the data into self.data

`write`(*fname*)

Try to write a file check we can write zipped also mimics that fabian was writing uint16 (we sometimes want floats)

4.14 `fabio.fit2dspreadsheetimage` Module

Read the fit2d ascii image output

- Jon Wright, ESRF

class `fabio.fit2dspreadsheetimage.`**`fit2dspreadsheetimage`**(*data=None*, *header=None*)

Bases: `fabio.fabioimage.fabioimage`

Read a fit2d ascii format

`read`(*fname*, *frame=None*)

Read in header into self.header and the data into self.data

4.15 `fabio.GEimage` Module

class `fabio.GEimage.`**`GEimage`**(*data=None*, *header=None*)

Bases: `fabio.fabioimage.fabioimage`

`getframe`(*num*)

Returns a frame as a new fabioimage object

`next`()

Get the next image in a series as a fabio image

`previous`()

Get the previous image in a series as a fabio image

`read`(*fname*, *frame=None*)

Read in header into self.header and the data into self.data

`write`(*fname*, *force_type=<type 'numpy.uint16'>*)

Not yet implemented

`fabio.GEimage.`**`demo`**()

4.16 `fabio.HiPiCimage` Module

Authors: Henning O. Sorensen & Erik Knudsen

Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk

- Jon Wright, ESRF

Information about the file format from Masakatzu Kobayashi is highly appreciated

class `fabio.HiPiCimage.`**`HiPiCimage`**(*data=None*, *header=None*)

Bases: `fabio.fabioimage.fabioimage`

Read HiPic images e.g. collected with a Hamamatsu CCD camera

`read`(*fname*, *frame=None*)

Read in header into self.header and the data into self.data

4.17 `fabio.kcdimage` Module

Authors: Jerome Kieffer, ESRF email:jerome.kieffer@esrf.fr

kcd images are 2D images written by the old KappaCCD diffractometer built by Nonius in the 1990's Based on the edfimage.py parser.

class `fabio.kcdimage.`**`kcdimage`**(*data=None*, *header=None*)

Bases: `fabio.fabioimage.fabioimage`

Read the Nonius kcd data format

static **`checkData`**(*data=None*)

`read`(*fname*, *frame=None*)

Read in header into self.header and the data into self.data

4.18 `fabio.mar345image` Module

4.18.1 Authors:

- Henning O. Sorensen & Erik Knudsen: Center for Fundamental Research: Metal Structures in Four Dimensions; Risoe National Laboratory; Frederiksborgvej 399; DK-4000 Roskilde; email:erik.knudsen@risoe.dk
- Jon Wright, Jérôme Kieffer & Gaël Goret: European Synchrotron Radiation Facility; Grenoble (France)

class `fabio.mar345image.`**`mar345image`**(**args*, ***kwargs*)

Bases: `fabio.fabioimage.fabioimage`

static **`checkData`**(*data=None*)

`nb_overflow_pixels`()

`read`(*fname*, *frame=None*)

Read a mar345 image

`write`(*fname*)

Try to write mar345 file. This is still in beta version. It uses CCP4 (LGPL) PCK1 algo from JPA

4.19 `fabio.marccdimage` Module

4.19.1 Authors:

- Henning O. Sorensen & Erik Knudsen: Center for Fundamental Research: Metal Structures in Four Dimensions; Risoe National Laboratory; Frederiksborgvej 399; DK-4000 Roskilde; email:erik.knudsen@risoe.dk
- Jon Wright: European Synchrotron Radiation Facility; Grenoble (France)

marccdimage can read MarCCD and MarMosaic images including header info.

JPW : Use a parser in case of typos (sorry?)

`fabio.marccdimage.`**`interpret_header`**(*header*, *fmt*, *names*)
given a format and header interpret it

`fabio.marccdimage.`**`make_format`**(*c_def_string*)
Reads the header definition in c and makes the format string to pass to struct.unpack

class `fabio.marccdimage.`**`marccdimage`**(**args*, ***kwds*)
Bases: `fabio.tifimage.tifimage`

Read in data in mar ccd format, also MarMosaic images, including header info

4.20 `fabio.OXDimage` Module

Reads Oxford Diffraction Sapphire 3 images

4.20.1 Authors:

- Henning O. Sorensen & Erik Knudsen: Center for Fundamental Research: Metal Structures in Four Dimensions; Risoe National Laboratory; Frederiksborgvej 399; DK-4000 Roskilde; email:erik.knudsen@risoe.dk
- Jon Wright, Jérôme Kieffer & Gaël Goret: European Synchrotron Radiation Facility; Grenoble (France)

class `fabio.OXDimage.`**`OXDimage`**(*data=None*, *header=None*)
Bases: `fabio.fabioimage.fabioimage`

Oxford Diffraction Sapphire 3 images reader/writer class

static **`checkData`**(*data=None*)

`getCompressionRatio`()
calculate the compression factor obtained vs raw data

`read`(*fname*, *frame=None*)
Read in header into self.header and the data into self.data

`write`(*fname*)
Write Oxford diffraction images: this is still beta :param fname: output filename

class `fabio.OXDimage.`**`Section`**(*size*, *dictHeader*)
Bases: `object`

Small helper class for writing binary headers

`getSize`(*dtype*)

`setData`(*key*, *offset*, *dtype*, *default=None*)

Parameters

- **offset** – int, starting position in the section
- **key** – name of the header key
- **dtype** – type of the data to insert (defines the size!)

4.21 `fabio.pilatusimage` Module

4.21.1 Authors:

- Henning O. Sorensen & Erik Knudsen: Center for Fundamental Research: Metal Structures in Four Dimensions; Risoe National Laboratory; Frederiksborgvej 399; DK-4000 Roskilde; email:erik.knudsen@risoe.dk
- Jon Wright: European Synchrotron Radiation Facility; Grenoble (France)

class `fabio.pilatusimage.`**`pilatusimage`**`(`**args*, ***kwds*`)`

Bases: `fabio.tifimage.tifimage`

Read in Pilatus format, also pilatus images, including header info

4.22 `fabio.pnmimage` Module

Authors: Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:henning.sorensen@risoe.dk

class `fabio.pnmimage.`**`pnmimage`**`(`**arg*, ***kwargs*`)`

Bases: `fabio.fabioimage.fabioimage`

static **`P1dec`**`(`*buf*, *bytecode*`)`

static **`P2dec`**`(`*buf*, *bytecode*`)`

static **`P3dec`**`(`*buf*, *bytecode*`)`

static **`P4dec`**`(`*buf*, *bytecode*`)`

static **`P5dec`**`(`*buf*, *bytecode*`)`

static **`P6dec`**`(`*buf*, *bytecode*`)`

static **`P7dec`**`(`*buf*, *bytecode*`)`

static **`checkData`**`(`*data=None*`)`

`read``(`*fname*, *frame=None*`)`

try to read PNM images :param fname: name of the file :param frame: not relevant here! PNM is always single framed

`write``(`*filename*`)`

4.23 `fabio.tifimage` Module

FabIO class for dealing with TIFF images. In facts wraps TiffIO from V. Armando Solé (available in PyMca) or falls back to PIL

4.23.1 Authors:

- Henning O. Sorensen & Erik Knudsen: Center for Fundamental Research: Metal Structures in Four Dimensions; Risoe National Laboratory; Frederiksborgvej 399; DK-4000 Roskilde; email:erik.knudsen@risoe.dk
- Jérôme Kieffer: European Synchrotron Radiation Facility; Grenoble (France)

License: GPLv3+

class `fabio.tifimage.`**`Image_File_Directory`**(*instring=None*, *offset=-1*)
Bases: `object`

`unpack`(*instring*, *offset=-1*)

class `fabio.tifimage.`**`Image_File_Directory_entry`**(*tag=0*, *tag_type=0*, *count=0*, *offset=0*)
Bases: `object`

`extract_data`(*full_string*)

`unpack`(*strInput*)

class `fabio.tifimage.`**`Tiff_header`**(*string*)
Bases: `object`

class `fabio.tifimage.`**`tifimage`**(**args*, ***kwds*)
Bases: `fabio.fabioimage.fabioimage`

Images in TIF format Wraps TiffIO

`read`(*fname*, *frame=None*)
Wrapper for TiffIO.

`write`(*fname*)
Overrides the fabioimage.write method and provides a simple TIFF image writer. :param fname: name of the file to save the image to @tag_type fname: string or unicode (file?)...

4.24 `fabio.xsdimage` Module

Authors: Jérôme Kieffer, ESRF email:jerome.kieffer@esrf.fr

XSDimge are XML files containing numpy arrays

class `fabio.xsdimage.`**`xsdimage`**(*data=None*, *header=None*, *fname=None*)
Bases: `fabio.fabioimage.fabioimage`

Read the XSDataImage XML File data format

`read`(*fname*, *frame=None*)

4.25 `fabio.compression` Module

Authors: Jérôme Kieffer, ESRF email:jerome.kieffer@esrf.fr

FabIO library containing compression and decompression algorithm for various

`fabio.compression.`**`compByteOffet_numpy`**(*data*)
Compress a dataset into a string using the byte_offet algorithm

Parameters **data** – ndarray

Returns string/bytes with compressed data

test = numpy.array([0,1,2,127,0,1,2,128,0,1,2,32767,0,1,2,32768,0,1,2,2147483647,0,1,2,2147483648,0,1,2,128,129,130,32767,3

`fabio.compression.`**`compPCK`**(*data*)

Modified CCP4 pck compressor used in MAR345 images

Parameters **data** – numpy.ndarray (square array)

Returns compressed stream

`fabio.compression.`**`compTY1`**(*data*)

Modified byte offset compressor used in Oxford Diffraction images

Parameters **data** – numpy.ndarray with the input data (integers!)

Returns 3-tuple of strings: raw_8,raw_16,raw_32 containing raw data with integer of the given size

`fabio.compression.`**`decByteOffet_cython`**(*stream*, *size=None*)

Analyze a stream of char with any length of exception: 2, 4, or 8 bytes integers

Parameters

- **stream** – string representing the compressed data
- **size** – the size of the output array (of longInts)

Returns 1D-ndarray

`fabio.compression.`**`decByteOffet_numpy`**(*stream*, *size=None*)

Analyze a stream of char with any length of exception: 2, 4, or 8 bytes integers

Parameters

- **stream** – string representing the compressed data
- **size** – the size of the output array (of longInts)

Returns 1D-ndarray

`fabio.compression.`**`decByteOffet_python`**(*stream*, *size*)

Analyze a stream of char with any length of exception (2,4, or 8 bytes integers)

Parameters

- **stream** – string representing the compressed data
- **size** – the size of the output array (of longInts)

Returns 1D-ndarray

`fabio.compression.`**`decByteOffet_weave`**(*stream*, *size*)

Analyze a stream of char with any length of exception (2,4, or 8 bytes integers)

Parameters

- **stream** – string representing the compressed data
- **size** – the size of the output array (of longInts)

Returns 1D-ndarray

`fabio.compression.`**`decBzip2`**(*stream*)

Decompress a chunk of data using the bzip2 algorithm from Python `fabio.compression.`**`decGzip`**`(`*stream*`)`

Decompress a chunk of data using the gzip algorithm from Python or alternatives if possible

`fabio.compression.`**`decKM4CCD`**`(`*raw_8*, *raw_16=None*, *raw_32=None*`)`

Modified byte offset decompressor used in Oxford Diffraction images

Parameters

- **raw_8** – strings containing raw data with integer 8 bits
- **raw_16** – strings containing raw data with integer 16 bits
- **raw_32** – strings containing raw data with integer 32 bits

Returns numpy.ndarray

`fabio.compression.`**`decPCK`**`(`*stream*, *dim1=None*, *dim2=None*, *overflowPix=None*, *version=None*`)`

Modified CCP4 pck decompressor used in MAR345 images

Parameters **stream** – string or file

Returns numpy.ndarray (square array)

`fabio.compression.`**`decTY1`**`(`*raw_8*, *raw_16=None*, *raw_32=None*`)`

Modified byte offset decompressor used in Oxford Diffraction images

Parameters

- **raw_8** – strings containing raw data with integer 8 bits
- **raw_16** – strings containing raw data with integer 16 bits
- **raw_32** – strings containing raw data with integer 32 bits

Returns numpy.ndarray

`fabio.compression.`**`decZlib`**`(`*stream*`)`

Decompress a chunk of data using the zlib algorithm from Python

`fabio.compression.`**`endianness`**`()`

Return the native endianness of the system

`fabio.compression.`**`md5sum`**`(`*blob*`)`

returns the md5sum of an object...

4.26 `fabio.converters` Module

Converter module. This is for the moment empty (populated only with almost pass through anonymous functions) but aims to be populated with more sofisticated translators ...

`fabio.converters.`**`convert_data`**`(`*inp*, *outp*, *data*`)`

Return data converted to the output format ... over-simplistic implementation for the moment ... :param inp,outp: input/output format like “cbfimage” :param data(ndarray): the actual dataset to be transformed

`fabio.converters.`**`convert_data_integer`**`(`*data*`)`

convert data to integer

`fabio.converters.`**`convert_header`**`(`*inp*, *outp*, *header*`)`

return header converted to the output format :param inp,outp: input/output format like “cbfimage” :param header(dict):the actual set of headers to be transformed

4.27 `fabio.datIO` Module

Authors: Henning O. Sorensen & Erik Knudsen Center for Fundamental Research: Metal Structures in Four Dimensions Risoe National Laboratory Frederiksborgvej 399 DK-4000 Roskilde email:erik.knudsen@risoe.dk

and Jon Wright, ESRF

class `fabio.datIO.`**`columnfile`**(*data=None*, *clabels=None*, *rlabels=None*, *fname=None*)

Bases: `fabio.datIO.fabiodata`

Concrete fabiodata class

`read`(*fname*, *frame=None*)

class `fabio.datIO.`**`fabiodata`**(*data=None*, *clabels=None*, *rlabels=None*, *fname=None*)

Bases: `object`

A common class for dataIO in fable Contains a 2d numpy array for keeping data, and two lists (clabels and rlabels) containing labels for columns and rows respectively

`read`(*fname=None*, *frame=None*)

To be overridden by format specific subclasses

4.28 `fabio.TiffIO` Module

class `fabio.TiffIO.`**`TiffIO`**(*filename*, *mode=None*, *cache_length=20*, *mono_output=False*)

Bases: `object`

`getData`(*nImage*, ***kw*)

`getImage`(*nImage*)

`getImageFileDirectories`(*fd=None*)

`getInfo`(*nImage*, ***kw*)

`getNumberOfImages`()

`writeImage`(*image0*, *info=None*, *software=None*, *date=None*)

4.29 `fabio.readbytestream` Module

Reads a bytestream

Authors: Jon Wright Henning O. Sorensen & Erik Knudsen ESRF Risoe National Laboratory

`fabio.readbytestream.`**`readbytestream`**(*fil*, *offset*, *x*, *y*, *nbytespp*, *datatype='int'*, *signed='n'*, *swap='n'*, *typeout=<type 'numpy.uint16'>*)

Reads in a bytestream from a file (which may be a string indicating a filename, or an already opened file (should be "rb")) offset is the position (in bytes) where the pixel data start nbytespp = number of bytes per pixel type can be int or float (4 bytes pp) or double (8 bytes pp) signed: normally signed data 'y', but 'n' to try to get back the right numbers when unsigned data are converted to signed (python once had no unsigned numeric types.) swap, normally do not bother, but 'y' to swap bytes typeout is the numpy type to output, normally uint16, but more if overflows occurred x and y are the pixel dimensions

TODO : Read in regions of interest

PLEASE LEAVE THE STRANGE INTERFACE ALONE - IT IS USEFUL FOR THE BRUKER FORMAT

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

f

fabio.__init__, 15
fabio.adscimage, 21
fabio.binaryimage, 22
fabio.bruker100image, 23
fabio.brukerimage, 23
fabio.cbfimage, 23
fabio.compression, 33
fabio.converters, 35
fabio.datIO, 36
fabio.dm3image, 25
fabio.edfimage, 25
fabio.fabioimage, 15
fabio.fabioutils, 17
fabio.file_series, 18
fabio.fit2dmaskimage, 29
fabio.fit2dspreadsheetimage, 29
fabio.GEimage, 29
fabio.HiPiCimage, 30
fabio.kcdimage, 30
fabio.mar345image, 30
fabio.marccdimage, 31
fabio.openimage, 21
fabio.OXDimage, 31
fabio.pilatusimage, 32
fabio.pnmimage, 32
fabio.readbytestream, 36
fabio.TiffIO, 36
fabio.tifimage, 32
fabio.xsdimage, 33

INDEX

A

add() (fabio.fabioimage.fabioimage method), 15
adscimage (class in fabio.adscimage), 21
appendFrame() (fabio.edfimage.edfimage method), 26

B

basic_translate() (fabio.brukerimage.brukerimage method), 23
BINARY_MARKER (fabio.cbfimage.CIF attribute), 24
binaryimage (class in fabio.binaryimage), 22
BLANK (fabio.cbfimage.CIF attribute), 24
bpp (fabio.edfimage.edfimage attribute), 26
bpp_to_numpy (fabio.brukerimage.brukerimage attribute), 23
bruker100image (class in fabio.bruker100image), 23
brukerimage (class in fabio.brukerimage), 23
bytecode (fabio.edfimage.edfimage attribute), 26
bytecode (fabio.edfimage.Frame attribute), 26
BZ2File (class in fabio.fabioutils), 17

C

calc_bpp() (fabio.brukerimage.brukerimage method), 23
capsHeader (fabio.edfimage.edfimage attribute), 26
cbfimage (class in fabio.cbfimage), 25
checkData() (fabio.cbfimage.cbfimage static method), 25
checkData() (fabio.fabioimage.fabioimage static method), 15
checkData() (fabio.fit2dmaskimage.fit2dmaskimage static method), 29
checkData() (fabio.kcdimage.kcdimage static method), 30
checkData() (fabio.mar345image.mar345image static method), 30
checkData() (fabio.OXDimage.OXDimage static method), 31
checkData() (fabio.pnmimage.pnmimage static method), 32
checkHeader() (fabio.edfimage.edfimage static method), 26
checkHeader() (fabio.fabioimage.fabioimage static method), 15
CIF (class in fabio.cbfimage), 24
classname (fabio.fabioimage.fabioimage attribute), 15
columnfile (class in fabio.datIO), 36
compByteOffet_numpy() (in module fabio.compression), 33
compPCK() (in module fabio.compression), 34
compTY1() (in module fabio.compression), 34
construct_filename() (in module fabio.fabioutils), 17
convert() (fabio.fabioimage.fabioimage method), 15
convert_data() (in module fabio.converters), 35
convert_data_integer() (in module fabio.converters), 35
convert_header() (in module fabio.converters), 35
current() (fabio.file_series.file_series method), 19
current() (fabio.file_series.filename_series method), 20
current_image() (fabio.file_series.file_series method), 19
current_image() (fabio.file_series.filename_series method), 20
current_object() (fabio.file_series.file_series method), 19
current_object() (fabio.file_series.filename_series method), 20

D

data (fabio.edfimage.edfimage attribute), 27
data (fabio.edfimage.Frame attribute), 26
decByteOffet_cython() (in module fabio.compression), 34
decByteOffet_numpy() (in module fabio.compression), 34
decByteOffet_python() (in module fabio.compression), 34
decByteOffet_weave() (in module fabio.compression), 34
decBzip2() (in module fabio.compression), 34
decGzip() (in module fabio.compression), 34
decKM4CCD() (in module fabio.compression), 35
deconstruct_filename() (fabio.fabioutils.FilenameObject method), 17
deconstruct_filename() (in module fabio.fabioutils), 18
decPCK() (in module fabio.compression), 35
decTY1() (in module fabio.compression), 35
decZlib() (in module fabio.compression), 35
delCapsHeader() (fabio.edfimage.edfimage method), 27
delData() (fabio.edfimage.edfimage method), 27

deleteFrame() (fabio.edfimage.edfimage method), 27
delHeader() (fabio.edfimage.edfimage method), 27
delHeaderKeys() (fabio.edfimage.edfimage method), 27
demo() (in module fabio.GEimage), 29
deprecated() (in module fabio.fabioutils), 18
dim1 (fabio.edfimage.edfimage attribute), 27
dim2 (fabio.edfimage.edfimage attribute), 27
dims (fabio.edfimage.edfimage attribute), 27
dm3image (class in fabio.dm3image), 25
do_magic() (in module fabio.openimage), 21

E

edfimage (class in fabio.edfimage), 26
endianness() (in module fabio.compression), 35
EOL (fabio.cbfimage.CIF attribute), 24
estimate_offset_value() (fabio.binaryimage.binaryimage method), 22
exists() (fabio.cbfimage.CIF method), 24
existsInLoop() (fabio.cbfimage.CIF method), 24
extract_data() (fabio.tifimage.Image_File_Directory_entry method), 33
extract_filenumber() (in module fabio.fabioutils), 18

F

fabio.__init__ (module), 15
fabio.adscimage (module), 21
fabio.binaryimage (module), 22
fabio.bruker100image (module), 23
fabio.brukerimage (module), 23
fabio.cbfimage (module), 23
fabio.compression (module), 33
fabio.converters (module), 35
fabio.datIO (module), 36
fabio.dm3image (module), 25
fabio.edfimage (module), 25
fabio.fabioimage (module), 15
fabio.fabioutils (module), 17
fabio.file_series (module), 18
fabio.fit2dmaskimage (module), 29
fabio.fit2dspreadsheetimage (module), 29
fabio.GEimage (module), 29
fabio.HiPiCimage (module), 30
fabio.kcdimage (module), 30
fabio.mar345image (module), 30
fabio.marccdimage (module), 31
fabio.openimage (module), 21
fabio.OXDimage (module), 31
fabio.pilatusimage (module), 32
fabio.pnmimage (module), 32
fabio.readbytestream (module), 36
fabio.TiffIO (module), 36
fabio.tifimage (module), 32
fabio.xsdimage (module), 33
fabiodata (class in fabio.datIO), 36
fabioimage (class in fabio.fabioimage), 15
fastReadData() (fabio.edfimage.edfimage method), 27
fastReadROI() (fabio.edfimage.edfimage method), 27
File (class in fabio.fabioutils), 17
file_series (class in fabio.file_series), 18
filename_series (class in fabio.file_series), 20
FilenameObject (class in fabio.fabioutils), 17
first() (fabio.file_series.file_series method), 19
first_image() (fabio.file_series.file_series method), 19
first_object() (fabio.file_series.file_series method), 19
fit2dmaskimage (class in fabio.fit2dmaskimage), 29
fit2dspreadsheetimage (class in fabio.fit2dspreadsheetimage), 29
Frame (class in fabio.edfimage), 26

G

GEimage (class in fabio.GEimage), 29
gen_header() (fabio.brukerimage.brukerimage method), 23
gen_overflow() (fabio.brukerimage.brukerimage method), 23
getBpp() (fabio.edfimage.edfimage method), 27
getByteCode() (fabio.edfimage.edfimage method), 27
getByteCode() (fabio.edfimage.Frame method), 26
getCapsHeader() (fabio.edfimage.edfimage method), 27
getclassname() (fabio.fabioimage.fabioimage method), 15
getCompressionRatio() (fabio.OXDimage.OXDimage method), 31
getData() (fabio.edfimage.edfimage method), 27
getData() (fabio.edfimage.Frame method), 26
getData() (fabio.TiffIO.TiffIO method), 36
getDim1() (fabio.edfimage.edfimage method), 27
getDim2() (fabio.edfimage.edfimage method), 27
getDims() (fabio.edfimage.edfimage method), 27
getEdfBlock() (fabio.edfimage.Frame method), 26
getframe() (fabio.edfimage.edfimage method), 27
getframe() (fabio.fabioimage.fabioimage method), 15
getframe() (fabio.GEimage.GEimage method), 29
getHeader() (fabio.edfimage.edfimage method), 27
getheader() (fabio.fabioimage.fabioimage method), 15
getHeaderKeys() (fabio.edfimage.edfimage method), 27
getImage() (fabio.TiffIO.TiffIO method), 36
getImageFileDirectories() (fabio.TiffIO.TiffIO method), 36
getInfo() (fabio.TiffIO.TiffIO method), 36
getmax() (fabio.fabioimage.fabioimage method), 15
getmean() (fabio.fabioimage.fabioimage method), 15
getmin() (fabio.fabioimage.fabioimage method), 16
getNbFrames() (fabio.edfimage.edfimage method), 27
getnum() (in module fabio.fabioutils), 18
getNumberOfImages() (fabio.TiffIO.TiffIO method), 36
getSize() (fabio.fabioutils.BZ2File method), 17
getSize() (fabio.fabioutils.File method), 17

getSize() (fabio.fabioutils.StringIO method), 17
getSize() (fabio.OXDimage.Section method), 31
getstddev() (fabio.fabioimage.fabioimage method), 16
GzipFile (class in fabio.fabioutils), 17

H

header (fabio.edfimage.edfimage attribute), 28
header_keys (fabio.edfimage.edfimage attribute), 28
HEADERS_KEYS (fabio.brukerimage.brukerimage attribute), 23
HiPiCimage (class in fabio.HiPiCimage), 30

I

Image_File_Directory (class in fabio.tifimage), 33
Image_File_Directory_entry (class in fabio.tifimage), 33
integrate_area() (fabio.fabioimage.fabioimage method), 16
interpret_header() (in module fabio.marccdimage), 31
isAscii() (fabio.cbfimage.CIF static method), 24
isAscii() (in module fabio.fabioutils), 18

J

jump() (fabio.file_series.file_series method), 19
jump() (fabio.file_series.filename_series method), 20
jump_filename() (in module fabio.fabioutils), 18
jump_image() (fabio.file_series.file_series method), 19
jump_image() (fabio.file_series.filename_series method), 20
jump_object() (fabio.file_series.file_series method), 19
jump_object() (fabio.file_series.filename_series method), 20

K

kcdimage (class in fabio.kcdimage), 30

L

last() (fabio.file_series.file_series method), 19
last_image() (fabio.file_series.file_series method), 19
last_object() (fabio.file_series.file_series method), 19
len() (fabio.file_series.file_series method), 19
load() (fabio.fabioimage.fabioimage method), 16
loadCHIPLOT() (fabio.cbfimage.CIF method), 24
loadCIF() (fabio.cbfimage.CIF method), 24
LoopHasKey() (fabio.cbfimage.CIF static method), 24

M

make_format() (in module fabio.marccdimage), 31
make_slice() (fabio.fabioimage.fabioimage method), 16
mar345image (class in fabio.mar345image), 30
marccdimage (class in fabio.marccdimage), 31
md5sum() (in module fabio.compression), 35

N

nb_overflow_pixels() (fabio.mar345image.mar345image method), 30
new_file_series() (in module fabio.file_series), 20
new_file_series0() (in module fabio.file_series), 21
next() (fabio.edfimage.edfimage method), 28
next() (fabio.fabioimage.fabioimage method), 16
next() (fabio.file_series.file_series method), 19
next() (fabio.file_series.filename_series method), 20
next() (fabio.GEimage.GEimage method), 29
next_filename() (in module fabio.fabioutils), 18
next_image() (fabio.file_series.file_series method), 19
next_image() (fabio.file_series.filename_series method), 20
next_object() (fabio.file_series.file_series method), 20
next_object() (fabio.file_series.filename_series method), 20
nframes (fabio.edfimage.edfimage attribute), 28
nice_int() (in module fabio.fabioutils), 18
numbered_file_series (class in fabio.file_series), 21
numstem() (in module fabio.fabioutils), 18

O

openheader() (in module fabio.openimage), 21
openimage() (in module fabio.openimage), 21
OXDimage (class in fabio.OXDimage), 31

P

P1dec() (fabio.pnmimage.pnmimage static method), 32
P2dec() (fabio.pnmimage.pnmimage static method), 32
P3dec() (fabio.pnmimage.pnmimage static method), 32
P4dec() (fabio.pnmimage.pnmimage static method), 32
P5dec() (fabio.pnmimage.pnmimage static method), 32
P6dec() (fabio.pnmimage.pnmimage static method), 32
P7dec() (fabio.pnmimage.pnmimage static method), 32
pad() (in module fabio.fabioutils), 18
parseheader() (fabio.edfimage.Frame method), 26
pilatusimage (class in fabio.pilatusimage), 32
pnmimage (class in fabio.pnmimage), 32
pop() (fabio.cbfimage.CIF method), 24
popitem() (fabio.cbfimage.CIF method), 24
prev_image() (fabio.file_series.filename_series method), 20
previous() (fabio.edfimage.edfimage method), 28
previous() (fabio.fabioimage.fabioimage method), 16
previous() (fabio.file_series.file_series method), 20
previous() (fabio.file_series.filename_series method), 20
previous() (fabio.GEimage.GEimage method), 29
previous_filename() (in module fabio.fabioutils), 18
previous_image() (fabio.file_series.file_series method), 20
previous_object() (fabio.file_series.file_series method), 20

previous_object() (fabio.file_series.filename_series method), 20

R

read() (fabio.adscimage.adscimage method), 22
read() (fabio.binaryimage.binaryimage method), 22
read() (fabio.bruker100image.bruker100image method), 23
read() (fabio.brukerimage.brukerimage method), 23
read() (fabio.cbfimage.cbfimage method), 25
read() (fabio.datIO.columnfile method), 36
read() (fabio.datIO.fabiodata method), 36
read() (fabio.dm3image.dm3image method), 25
read() (fabio.edfimage.edfimage method), 28
read() (fabio.fabioimage.fabioimage method), 16
read() (fabio.fit2dmaskimage.fit2dmaskimage method), 29
read() (fabio.fit2dspreadsheetimage.fit2dspreadsheetimage method), 29
read() (fabio.GEimage.GEimage method), 29
read() (fabio.HiPiCimage.HiPiCimage method), 30
read() (fabio.kcdimage.kcdimage method), 30
read() (fabio.mar345image.mar345image method), 30
read() (fabio.OXDimage.OXDimage method), 31
read() (fabio.pnmimage.pnmimage method), 32
read() (fabio.tifimage.tifimage method), 33
read() (fabio.xsdimage.xsdimage method), 33
read_data() (fabio.dm3image.dm3image method), 25
read_tag_entry() (fabio.dm3image.dm3image method), 25
read_tag_group() (fabio.dm3image.dm3image method), 25
read_tag_type() (fabio.dm3image.dm3image method), 25
readbytes() (fabio.dm3image.dm3image method), 25
readbytestream() (in module fabio.readbytestream), 36
readCIF() (fabio.cbfimage.CIF method), 24
readheader() (fabio.fabioimage.fabioimage method), 16
readROI() (fabio.fabioimage.fabioimage method), 16
rebin() (fabio.fabioimage.fabioimage method), 16
resetvals() (fabio.fabioimage.fabioimage method), 16

S

save() (fabio.fabioimage.fabioimage method), 16
saveCIF() (fabio.cbfimage.CIF method), 24
Section (class in fabio.OXDimage), 31
setBpp() (fabio.edfimage.edfimage method), 28
setByteCode() (fabio.edfimage.edfimage method), 28
setByteCode() (fabio.edfimage.Frame method), 26
setCapsHeader() (fabio.edfimage.edfimage method), 28
setData() (fabio.edfimage.edfimage method), 28
setData() (fabio.edfimage.Frame method), 26
setData() (fabio.OXDimage.Section method), 31
setDim1() (fabio.edfimage.edfimage method), 28
setDim2() (fabio.edfimage.edfimage method), 28
setHeader() (fabio.edfimage.edfimage method), 28
setHeaderKeys() (fabio.edfimage.edfimage method), 28
setNbFrames() (fabio.edfimage.edfimage method), 28
setSize() (fabio.fabioutils.BZ2File method), 17
setSize() (fabio.fabioutils.File method), 17
setSize() (fabio.fabioutils.StringIO method), 17
size (fabio.fabioutils.BZ2File attribute), 17
size (fabio.fabioutils.File attribute), 17
size (fabio.fabioutils.StringIO attribute), 17
SPACER (fabio.brukerimage.brukerimage attribute), 23
START_COMMENT (fabio.cbfimage.CIF attribute), 24
str() (fabio.fabioutils.FilenameObject method), 17
StringIO (class in fabio.fabioutils), 17
swap_needed() (fabio.binaryimage.binaryimage static method), 22
swap_needed() (fabio.edfimage.edfimage method), 28
swap_needed() (fabio.edfimage.Frame method), 26

T

test() (in module fabio.adscimage), 22
test() (in module fabio.brukerimage), 23
test() (in module fabio.fabioimage), 16
Tiff_header (class in fabio.tifimage), 33
TiffIO (class in fabio.TiffIO), 36
tifimage (class in fabio.tifimage), 33
toAscii() (in module fabio.fabioutils), 18
toPIL16() (fabio.bruker100image.bruker100image method), 23
toPIL16() (fabio.fabioimage.fabioimage method), 16
tostring() (fabio.cbfimage.CIF method), 25
tostring() (fabio.fabioutils.FilenameObject method), 17

U

UnknownCompressedFile (class in fabio.fabioutils), 17
unpack() (fabio.edfimage.edfimage method), 28
unpack() (fabio.tifimage.Image_File_Directory method), 33
unpack() (fabio.tifimage.Image_File_Directory_entry method), 33
update_header() (fabio.fabioimage.fabioimage method), 16

W

write() (fabio.adscimage.adscimage method), 22
write() (fabio.binaryimage.binaryimage method), 22
write() (fabio.brukerimage.brukerimage method), 23
write() (fabio.cbfimage.cbfimage method), 25
write() (fabio.edfimage.edfimage method), 28
write() (fabio.fabioimage.fabioimage method), 16
write() (fabio.fit2dmaskimage.fit2dmaskimage method), 29
write() (fabio.GEimage.GEimage method), 29
write() (fabio.mar345image.mar345image method), 30
write() (fabio.OXDimage.OXDimage method), 31

write() (fabio.pnmimage.pnmimage method), 32
write() (fabio.tifimage.tifimage method), 33
writeImage() (fabio.TiffIO.TiffIO method), 36

X

xsdimage (class in fabio.xsdimage), 33