FablO Documentation

Release 0.1.2

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

March 15, 2013

CONTENTS

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 format). 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 (Table format 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 Installation and usage

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
```

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 {tub05}and the neutron sources ISIS, SNS and SINQ already use HDF extensively through the NeXus {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 {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 {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.}

Python Module & Detector / Format & Extension & Read & Multi-image & Write

ADSC & ADSC Quantum & .img & Yes& - & Yes Bruker & Bruker formats & .sfrm & Yes& - &Yes DM3 & Gatan Digital Micrograph & .dm3 & Yes& - & - EDF & ESRF data format & .edf & Yes& Yes & Yes EDNA-XML & Used by EDNA {edna}& .xml & Yes& - & - CBF & CIF binary files & .cbf & Yes& - & Yes kcd & Nonius KappaCCD & .kccd & Yes& - & - fit2dmask & Used by Fit2D {fit2d}& .msk & Yes& - & Yes fit2dspreadsheet & Used by Fit2D {fit2d}& .spr & Yes& -& Yes GE & General Electric & - & Yes& Yes & - HiPiC & Hamamatsu CCD & .tif & Yes& - & - marccd & MarCCD/Mar165 & .mccd & Yes& - &Yes mar345 & Mar345 image plate & .mar3450 & Yes& - & Yes OXD & Oxford Diffraction & .img & Yes& - & Yes pilatus & Dectris Pilatus Tiff & .tif & Yes& - & Yes PNM & Portable aNy Map & .pnm & Yes& - & - TIFF & Tagged Image File Format & .tif & Yes& - & Yes

CHAPTER

TWO

FABIO PACKAGE

2.1 fabio Package

2.2 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

2.3 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.**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

closed

getSize()

seek(*offset*, *whence=0*)
Move to new file position.

Argument offset is a byte count. Optional argument whence defaults to 0 (offset from start of file, offset should be >= 0); other values are 1 (move relative to current position, positive or negative), and 2 (move relative to end of file, usually negative, although many platforms allow seeking beyond the end of a file). If the file is opened in text mode, only offsets returned by tell() are legal. Use of other offsets causes undefined behavior.

This is a wrapper for seek to ensure compatibility with old python 2.5

setSize(*value*)

size

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*)
Try to construct the filename for a given frame

fabio.fabioutils.**deconstruct_filename**(*filename*)
Break up a filename to get image type and number

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

class fabio.fabioutils.**filename_object**(*stem*, *num=None*, *directory=None*, *format=None*, *extension=None*, *postnum=None*, *digits=4*)
The 'meaning' of a filename

str()
Return a string representation

tostring()
convert yourself to a string

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.**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

2.4 fabio.file_series 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.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

current_object()
current image in sequence

first()
first image in series

first_image()
first image in a sequence

first_object()
first image in a sequence

jump(*num*)
goto a position in sequence

jump_image(*num*)
jump to and read image

jump_object(*num*)
jump to and read image

last()
last in series

last_image()
last image in a sequence

last_object()
last image in a sequence

len()
number of files

next()
next in a sequence

next_image()
Return the next image

next_object()
Return the next image

previous()
prev in a sequence

previous_image()
Return the previous image

previous_object()
Return the previous image

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.filename_object

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.filename_object

next()
increment number

next_image()
returns the next image as a fabioimage

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

prev_image()
returns the previos image as a fabioimage

previous()
decrement number

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

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.

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"

2.5 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

2.6 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

2.7 fabio.binaryimage Module

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

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

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, dim1, dim2, offset, bytecode, endian fname : file name : str dim1,dim2 : image dimensions : int offset : size of the : int bytecode among : "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*)

2.8 fabio.bruker100image Module

class fabio.bruker100image.**bruker100image**(*data=None*, *header=None*)
Bases: fabio.brukerimage.brukerimage

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

toPIL16(*filename=None*)

2.9 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

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

Read and eventually write ID11 bruker (eg smart6500) images

read(*fname*, *frame=None*)
Read in and unpack the pixels (including overflow table

write(*fname*)
Writes the image as EDF FIXME - this should call edfimage.write if that is wanted? eg: obj = edfimage(data = self.data, header = self.header)

obj.write(fname) or maybe something like: edfimage.write(self, fname)

write2(*fname*)
FIXME: what is this?

fabio.brukerimage.**test**()
a testcase

2.10 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.

static LoopHasKey(*loop*, *key*)
Returns True if the key (string) exist in the array called loop

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 :param _strFilename: the name of the filename to be appended in the

header of the CIF file

:return : a sting that corresponds to the content of the CIF - file. :rtype: string

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

2.11 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*)

2.12 fabio.edfimage Module

License: GPLv2+

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

2011-02-11: Mostly rewritten by Jérôme Kieffer (Jerome.Kieffer@esrf.eu) European Synchrotron Radiation Facility Grenoble (France)

2012-08-20: laisy read of data in EDF

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

2.13 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)

2.14 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

2.15 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**()

2.16 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

2.17 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

2.18 fabio.mar345image 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, Jerome Kieffer, Gael Goret ESRF, 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

2.19 fabio.marccdimage 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) Jon Wright, ESRF

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

2.20 fabio.OXDimage Module

Reads Oxford Diffraction Sapphire 3 images

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
- Gaël Goret, ESRF
- Jérôme Kieffer, ESRF

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!)

2.21 fabio.pilatusimage 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) Jon Wright, ESRF

marccdimage can read MarCCD and MarMosaic images including header info.

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

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

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

2.22 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*)

2.23 fabio.tifimage Module

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

Authors: Jérôme Kieffer (jerome.kieffer@esrf.fr)

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.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?)...

2.24 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*)

2.25 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 :param data: ndarray :return: 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

:param data numpy.ndarray (square array) :return compressed stream

fabio.compression.**compTY1**(*data*)
Modified byte offset compressor used in Oxford Diffraction images

:param data numpy.ndarray :return raw_8,raw_16,raw_32: strings 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

:return list of NParrays

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

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

:return list of NParrays

fabio.compression.**decByteOffet_python**(*stream*, *size*)
Analyze a stream of char with any length of exception (2,4, or 8 bytes integers) :param stream: string representing the compressed data :param size: the size of the output array (of longInts) :return :NParrays

fabio.compression.**decByteOffet_weave**(*stream*, *size*)
Analyze a stream of char with any length of exception (2,4, or 8 bytes integers)

:return list of NParrays

fabio.compression.**decBzip2**(*stream*)
decompress a chunk of data using the bzip2 algorithm

fabio.compression.**decGzip**(*stream*)

fabio.compression.**decKM4CCD**(*raw_8*, *raw_16=None*, *raw_32=None*)
Modified byte offset decompressor used in Oxford Diffraction images :param raw_8,raw_16,raw_32: strings containing raw data with integer of the given size :return numpy.ndarray

fabio.compression.**decPCK**(*stream*, *dim1=None*, *dim2=None*, *overflowPix=None*)
Modified CCP4 pck decompressor used in MAR345 images

Parameters **stream** – string or file

:return numpy.ndarray (square array)

fabio.compression.**decTY1**(*raw_8*, *raw_16=None*, *raw_32=None*)
Modified byte offset decompressor used in Oxford Diffraction images :param raw_8,raw_16,raw_32: strings containing raw data with integer of the given size :return numpy.ndarray

fabio.compression.**decZlib**(*stream*)
decompress a chunk of data using the zlib algorithm

fabio.compression.**endianness**()

fabio.compression.**md5sum**(*blob*)
returns the md5sum of an object...

2.26 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

2.27 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

2.28 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*)

2.29 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

THREE

INDICES AND TABLES

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

PYTHON MODULE INDEX

f

INDEX

J

K

L

M

N

O

P

R

S

T

U

W

X