FabIO

easy access to two-dimensional X-ray detector images in Python

Bergbäck Knudsen, Erik; Sørensen, Henning O.; Wright, Jonathan P.; Goret, Gael; Kieffer, Jerome

Published in:
Journal of Applied Crystallography

Link to article, DOI:
10.1107/S0021889813000150

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

Journal of
Applied Crystallography

ISSN 0021-8898

Received 14 September 2012
Accepted 2 January 2013

FabIO: easy access to two-dimensional X-ray detector images in Python

Erik B. Knudsen,[a] Henning O. Sørensen,[b] Jonathan P. Wright,[c] Gaël Goret[c] and Jérôme Kieffer[c]*

[a]Department of Physics, Technical University of Denmark, Kongens Lyngby, Denmark, [b]Nano-Science Center, Department of Chemistry, University of Copenhagen, Universitetsparken 5, Copenhagen, Denmark, and [c]European Synchrotron Radiation Facility, Grenoble, France. Correspondence e-mail: jerome.kieffer@esrf.fr

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` objects offer additional methods to extract information about the image and to open other detector images from the same data series.

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* image viewer (Sørensen & Knudsen, 2006) and *ImageD11* (Wright, 2005) peak-search programs and has been developed since 2007 as part of the *TotalCryst* (Poulsen *et al.*, 2006) program suite for analysis of three-dimensional X-ray diffraction microscopy data (Sørensen *et al.*, 2012). During integration into a range of scientific programs such as the *Fable* graphical interface (Götz *et al.*, 2007), *EDNA* (Incardona *et al.*, 2009) and the fast azimuthal integration library pyFAI (Kieffer & Karkoulis, 2013), FabIO has gained several features, including the ability to handle multi-frame image formats as well as to write many of the file formats.

We believe FabIO is now ready for a wider audience and could save other researchers from repeating the work involved in decoding a binary file format. Table 1 shows the list of file formats that FabIO can currently (version 0.1.0) read.

2. FabIO Python module

Python (van Rossum, 1989) is a scripting language that is very popular among scientists and which also allows well structured applications and libraries to be developed.

2.1. Philosophy

The intention behind this development was to create a Python module that would enable easy reading of two-dimensional 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 (Deutsch, 1996) and bzip2 (Burrows & Wheeler, 1994) compression transparently. Opening a file returns an object that stores the image data in memory as a two-dimensional NumPy array (Oliphant, 2007) and the metadata, called header, in a Python dictionary. Besides 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 access the 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 online (Knudsen *et al.*, 2007), licensed under the GNU General Public License version 3. FabIO is also available directly from popular Linux distributions like Debian and Ubuntu.

2.2. Implementation

The main language used in the development of FabIO is Python (van Rossum, 1989); 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 for example standard C code callable from Python (sometimes generated using Cython; Behnel *et al.*, 2011). 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 (Behnel *et al.*, 2005) is required, and the Python Image Library (PIL; PythonWare, 2005) 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 tools are also available in the scipy.ndimage module (Jones *et al.*, 2001) and in scikits-image (van der Walt, 2011).

Images can also be displayed in a convenient interactive manner using matplotlib (Hunter, 2007) and an IPython shell (Pérez & Granger, 2007), which is mainly used for developing data analysis algorithms. The reading and writing procedure of the various TIFF (ISO, 2004) formats is based on the *TiffIO* code from *PyMCA* (Solé *et al.*, 2007).

Table 1
List of file formats that FabIO can read and write (in alphabetical order).

The listed file name 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	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* (Incardona *et al.*, 2009)	`.xml`	Yes	No	No
CBF	CIF binary files	`.cbf`	Yes	No	Yes
kcd	Nonius KappaCCD	`.kccd`	Yes	No	No
fit2dmask	Used by *Fit2D* (Hammersley, 1997)	`.msk`	Yes	No	Yes
fit2dspreadsheet	Used by *Fit2D* (Hammersley, 1997)	`.spr`	Yes	No	Yes
GE	General Electric	–	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

In the Python shell, the FabIO module must be imported prior to reading an image in one of the supported file formats (see Table 1). 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 two-dimensional 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 is 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) is 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 1 summarizes writable formats). Automatic translation between file formats, even if desirable, is sometimes impossible because not all formats 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.

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 that return the file number, name *etc.* Some of the most important methods are specific to 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 a single-frame format (like mar345), `img.next()` will return the image in the next file; for a multi-frame format (like GE), `img.next()` will return the next frame within the same file. For formats that are possibly multi-framed like EDF, the behaviour depends on the actual number of frames per file (accessible *via* the `img.nframes` attribute).

3. Installation and usage

FabIO can, as any Python module, be installed from its sources, available on sourceforge (Knudsen *et al.*, 2007), but we recommend using the 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`.

3.1. Examples

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

(1) Opening an image:

```
import fabio
im100 = fabio.open('Quartz_0100.tif') # Open image file
print(im000.data[1024,1024])          # Check a pixel value
im101 = im100.next()                  # Open next image
im270 = im100.getframe(270)           # Jump to file number 270:
                                      # Quartz_0270.tif
```

(2) Normalizing 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',
...
}
# Normalize to beam current and save data
srcur = float(img.header['ESRFCurrent'])
img.data *= 200.0/srcur
img.write('normed_0001.edf')
```

(3) Interactive viewing with matplotlib:

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

4. Future and perspectives

The Hierarchical Data Format version 5 (HDF Group, 2010) is a data format that is increasingly popular for storage of X-ray and neutron data. To name a few facilities, the synchrotron Soleil (Poirier *et al.*, 2009) and the neutron sources ISIS, SNS and SINQ already use HDF extensively through the NeXus (NIAC, 2012) 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 (Collette, 2008), 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 of FabIO as string and file handling have been altered. This change is already ongoing as many parts of the native code in C have already been translated into Cython (Behnel *et al.*, 2011) to smooth the transition, since Cython generates code compatible with Python3. This also makes it easier to retain backwards compatibility with the earlier Python versions.

5. Conclusion

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

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).

References

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S. & Smith, K. (2011). *J. Comput. Sci. Eng.* **13**, 31–39.
Behnel, S., Faassen, M. *et al.* (2005). *lxml: XML and HTML with Python*, http://lxml.de.
Burrows, M. & Wheeler, D. (1994). SRC Research Report 124. Systems Research Center, Digital Equipment Corporation, Palo Alto, CA, USA, http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.
Collette, A. (2008). *HDF5 for Python*, http://h5py.alfven.org.
Deutsch, P. (1996). *GZIP File Format Specification Version 4.3*, http://tools.ietf.org/html/rfc1952.
Götz, A., Suchet, G. & Evan, K. (2007). *Fable User Interface*. ESRF, Grenoble, France, http://fable.sf.net.
Hammersley, A. P. (1997). *FIT2D: An Introduction and Overview*. ESRF Internal Report. ESRF, Grenoble, France.
HDF Group (2010). *Hierarchical Data Format Version 5*, http://www.hdfgroup.org/HDF5.
Hunter, J. D. (2007). *Comput. Sci. Eng.* **9**, 90–95.
Incardona, M.-F., Bourenkov, G. P., Levik, K., Pieritz, R. A., Popov, A. N. & Svensson, O. (2009). *J. Synchrotron Rad.* **16**, 872–879.
ISO (2004). *Graphic Technology – Prepress Digital Data Exchange – Tag Image File Format for Image Technology*. ISO 12639:2004. ISO, Geneva, Switzerland.
Jones, E., Oliphant, T. E. & Peterson, P. (2001). *SciPy*, http://www.scipy.org/.
Kieffer, J. & Karkoulis, D. (2013). *J. Phys. Conf. Ser.* In the press.
Knudsen, E. B., Sørensen, H. O., Wright, J. P., Goret, G. & Kieffer, J. (2007). *FabIO*, http://sourceforge.net/projects/fable/files/fabio.
NIAC (2012). *NeXus: A Common Data Format for Neutron, X-ray and Muon Science*. NeXus International Advisory Committee, http://www.nexusformat.org.
Oliphant, T. E. (2007). *Comput. Sci. Eng.* **9**, 10–20.
Pérez, F. & Granger, B. E. (2007). *Comput. Sci. Eng.* **9**, 21–29.
Poirier, S., Buteau, A., Gagey, B., Martinez, P., Maréchal, C., Mederbel, M., Ounsy, M., Pierrot, P. & Rochat, J. M. (2009). *Proceedings of ICALEPCS2009*, http://accelconf.web.cern.ch/accelconf/icalepcs2009/papers/tub005.pdf.
Poulsen, H. F. *et al.* (2006). *TotalCryst*, http://www.totalcryst.dk/.
PythonWare (2005). *Python Imaging Library (PIL)*, http://www.pythonware.com/products/pil/.
Rossum, G. van (1989). *Python Programming Language*, http://www.python.org.
Solé, V. A., Papillon, E., Cotte, M., Walter, Ph. & Susini, J. (2007). *Spectrochim. Acta Part B*, **62**, 63–68.
Sørensen, H. O. & Knudsen, E. B. (2006). *Fabian*. Risø National Laboratory for Sustainable Development, Technical University of Denmark, Roskilde, Denmark, http://fable.svn.sourceforge.net/svnroot/fable/fabian.
Sørensen, H. O., Schmidt, S., Wright, J. P., Vaughan, G. B. M., Techert, S., Garman, E. F., Oddershede, J., Davaasambuu, J., Paithankar, K. S., Gundlach, C. & Poulsen, H. F. (2012). *Z. Kristallogr.* **227**, 63–78.
Walt, S. van der (2011). *scikits-image*, http://scikits-image.org/.
Wright, J. P. (2005). *ImageD11*. ESRF, Grenoble, France, http://fable.svn.sourceforge.net/svnroot/fable/ImageD11.