Xfluo Documentation

Release 0.0.1

Argonne National Laboratory

May 04, 2019

Contents

This is a template to add sphinx / Read The Docs documentation to any python project to generate this Xfluo Docs.

These pages are written using reStructuredText that allows *emphasis*, **strong**, `literal` and many more styles.

You can add a reference [A1], include equations like:

$$V(x) = \left(\frac{1-\eta}{\sigma\sqrt{2\pi}}\right) \cdot exp\left(\frac{x^2}{2\sigma^2}\right) + \eta \cdot \frac{\sigma}{2\pi} \cdot \frac{1}{x^2 + \left(\frac{\sigma}{2}\right)^2}$$

or

$$I_{white} = \int_{E_1}^{E_2} I(\theta, E) \cdot F(E)\, dE.$$

and tables:

Member	Type	Example
first	ordinal	1st
second	ordinal	2nd
third	ordinal	3rd

More examples:

$$X(e^{j\omega}) = x(n)e^{-j\omega n}$$

Warning: Warning text.

Note: Note text.

CHAPTER 1

Features

- List here
- the module features

CHAPTER 2

Contribute

- Documentation: https://github.com/tomography/xfluo/tree/master/doc
- Issue Tracker: https://github.com/tomography/xfluo/docs/issues
- Source Code: https://github.com/tomography/xfluo/xfluo

CHAPTER 3

Content

3.1 About

This section describes what the Xfluo is about.

3.2 Install

This section covers the basics of how to download and install Xfluo. We recommend you to install the Anaconda Python distribution.

Contents:

3.2.1 Installing from source

Clone the Xfluo from GitHub repository:

```
git clone https://github.com/tomography/xfluo.git xfluo
```

then:

```
cd xfluo
python setup.py install
```

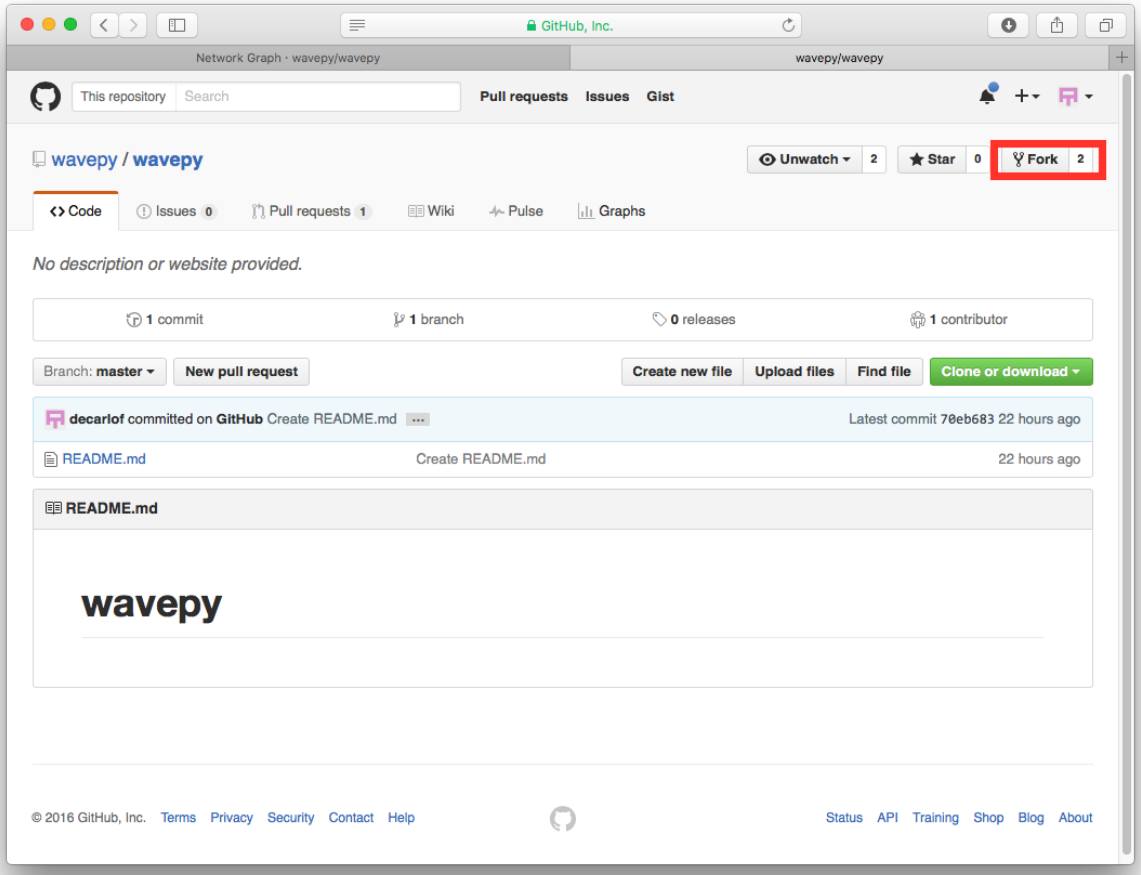
3.3 Development

This section explains the basics for developers who wish to contribute to a project mantained on GitHub and using the fork / pull request mechanism for accepting developer contributions.

Contents:

3.3.1 Fork the repository

The project is maintained on GitHub, which is a version control and a collaboration platform for software developers. To start first register on GitHub and fork the Xfluo repository by clicking the **Fork** button in the header of the Xfluo repository:

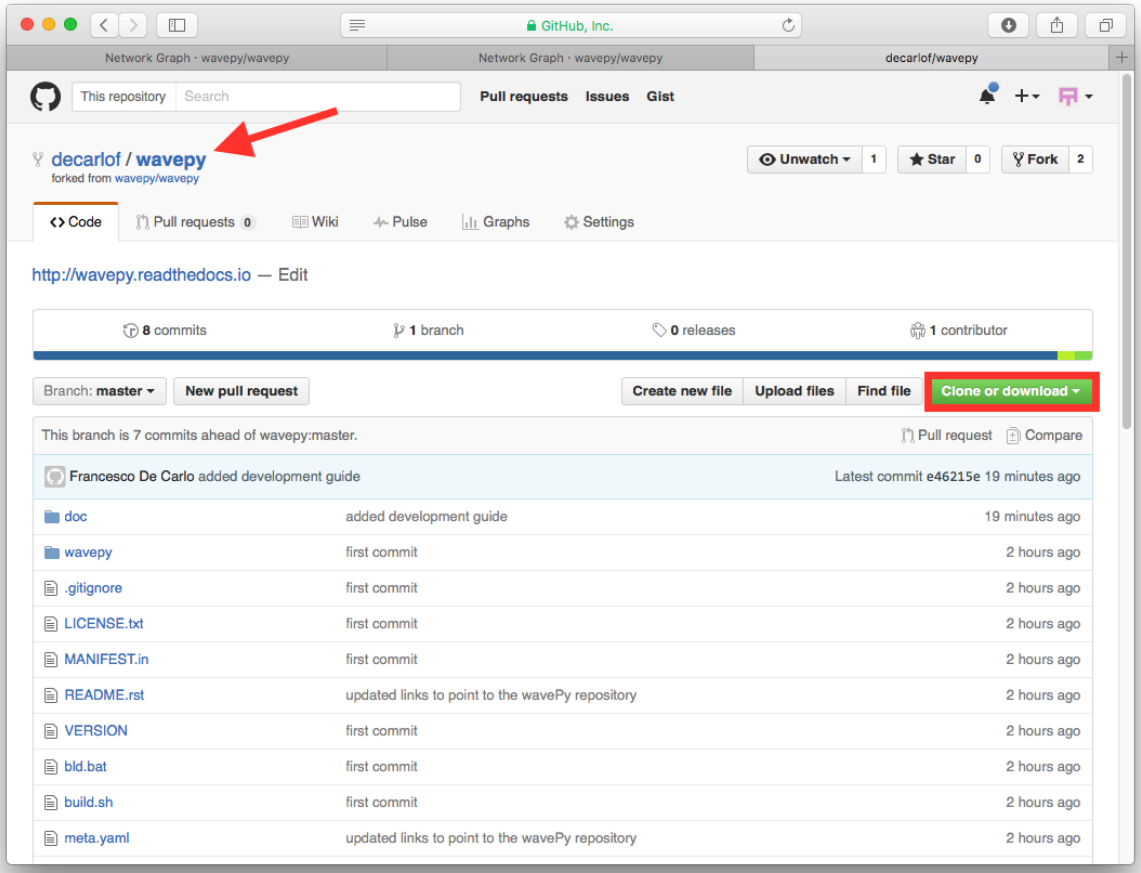


This successfully creates a copy of the project in your personal GitHub space.

3.3.2 Clone the repository

The next thing you want to do is to clone the repository you just created in your personal GitHub space to your local machine.

You can do this by clicking the **Clone in Desktop** button in the bottom of the right hand side bar:



This will launch the GitHub desktop application (available for both Mac and Win) and ask you where you want to save it. Select a location in your computer and feel comfortable with making modifications in the code.

3.3.3 Coding conventions

We try to keep a consistent and readable code. So, please keep in mind the following style and syntax guidance before you start coding.

First of all the code should be well documented, easy to understand, and integrate well into the rest of the project. For example, when you are writing a new function always describe the purpose and the parameters:

```
def my_awesome_func(a, b):
    """
    Adds two numbers.

    Parameters
    ----------
    a : scalar (float)
        First number to add

    b : scalar (float)
        Second number to add

```

(continues on next page)

(continued from previous page)

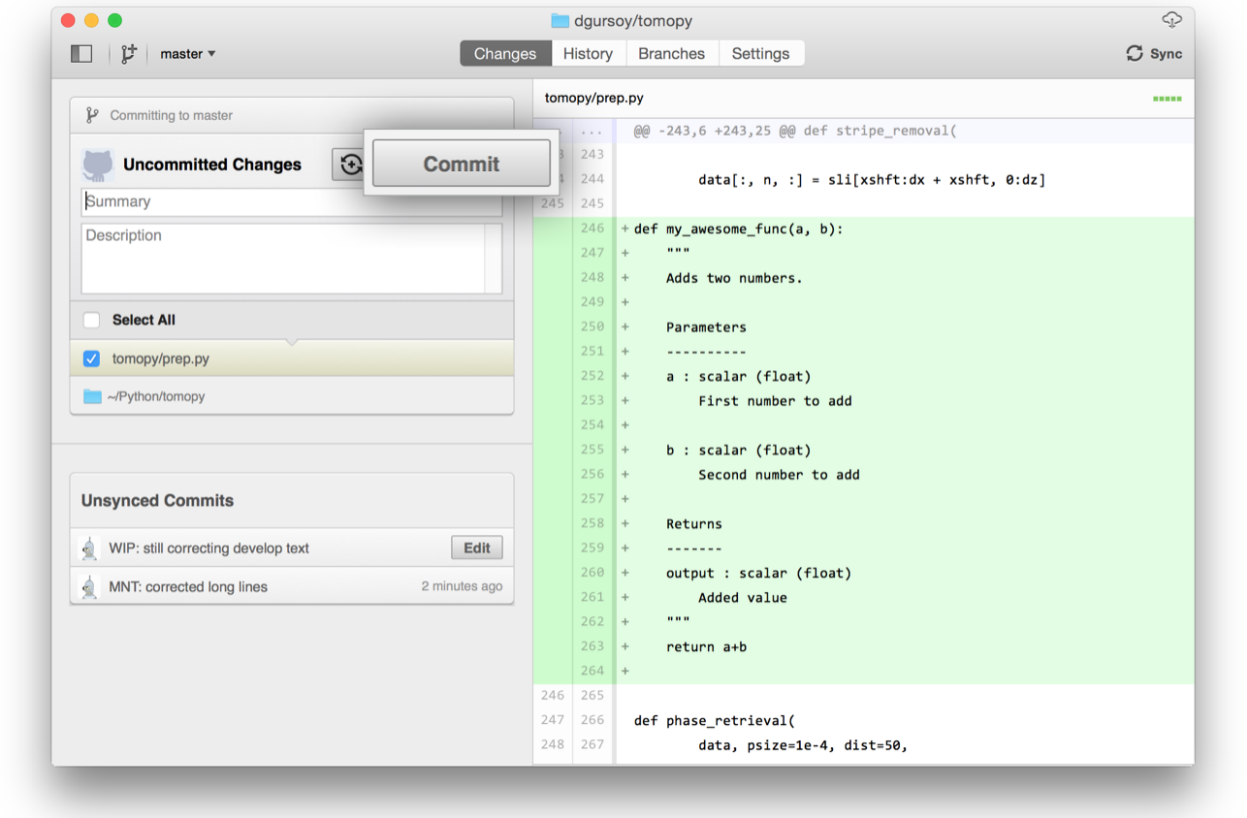
```
    Returns
    -------
    output : scalar (float)
        Added value
    """
    return a+b
```

3.3.4 Package versioning

We follow the X.Y.Z (Major.Minor.Patch) semantic for package versioning. The version should be updated before each pull request accordingly. The patch number is incremented for minor changes and bug fixes which do not change the software's API. The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

3.3.5 Commiting changes

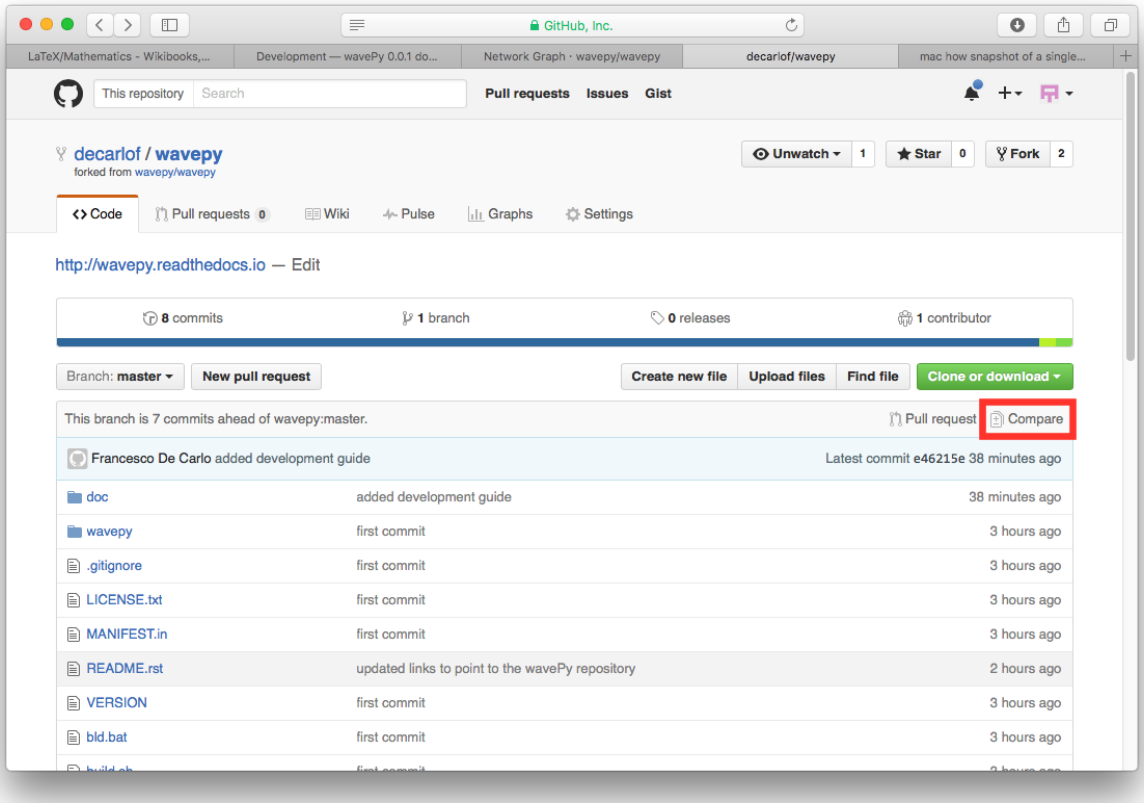
After making some changes in the code, you may want to take a *snapshot* of the edits you made. That's when you make a *commit*. To do this, launch the GitHub desktop application and it should provide you all the changes in your code since your last commit. Write a brief *Summary* and *Description* about the changes you made and click the **Commit** button:



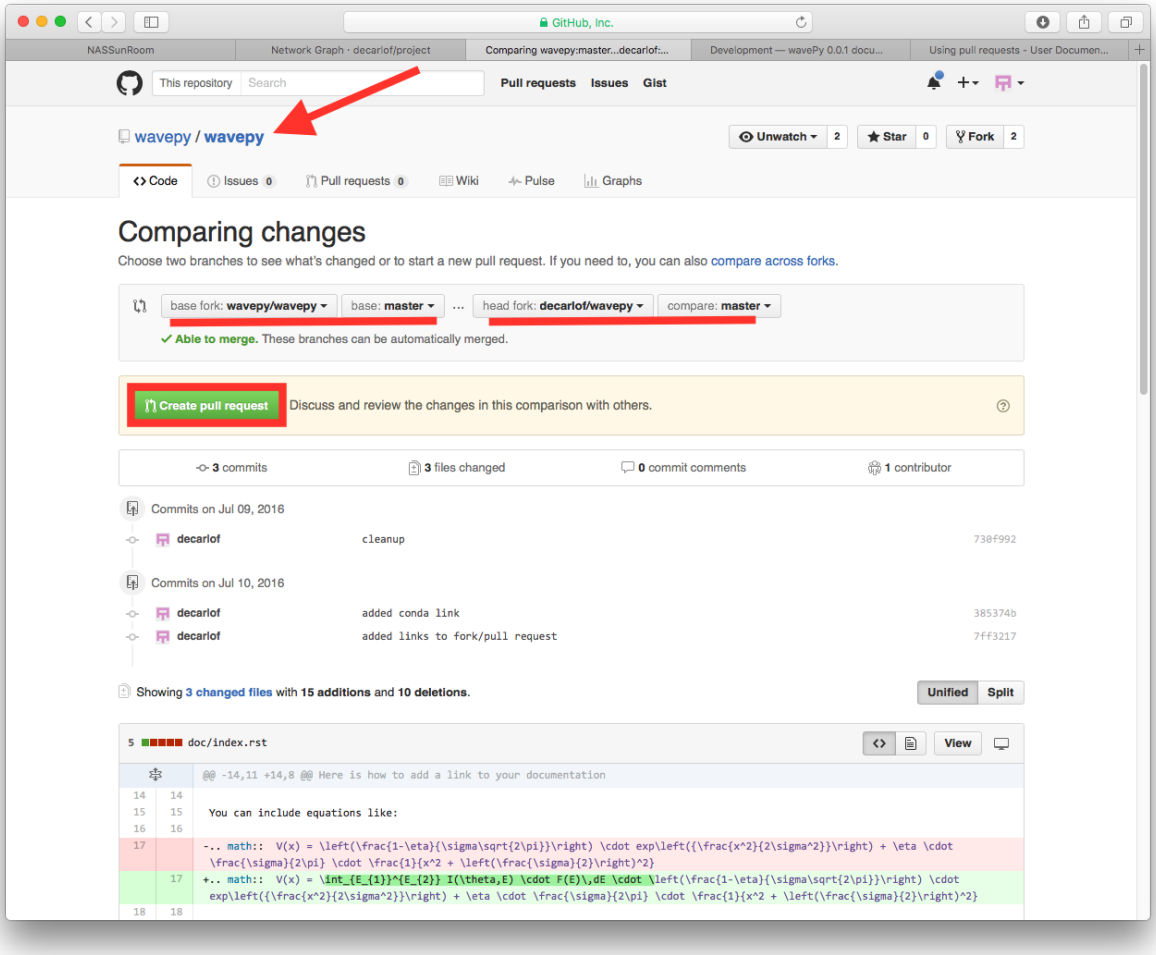
You can continue to make changes, add modules, write your own functions, and take more *Commit snapshots* of your code writing process.

3.3.6 Contributing back

Once you feel that the functionality you added would benefit the community, then you should consider contributing back to the project. For this, go to your online GitHub repository of the project and click on the *compare* button to compare, review and create a pull request.



After clicking on this button, you are presented with a review page where you can get a high-level overview of what exactly has changed between your forked branch and the original project repository. When you're ready to submit your pull request, click **Create pull request**:



Clicking on **Create pull request** sends you to a discussion page, where you can enter a title and optional description. It's important to provide as much useful information and a rationale for why you're making this Pull Request in the first place.

When you're ready typing out your heartfelt argument, click on **Send pull request**.

You're done!

3.4 API reference

Xfluo Modules:

3.4.1 `xfluo.reco`

Functions:

tomo(params)

`xfluo.reco.`**`tomo`**(*params*)

3.4.2 xfluo.elements

Properties of the chemical elements.

Each chemical element is represented as an object instance. Physicochemical and descriptive properties of the elements are stored as instance attributes.

Author Christoph Gohlke

Version 2019.1.1

Requirements

- CPython 2.7 or 3.5+

Revisions

2019.1.1 Update copyright year.

2018.8.15 Move modules into molmass package.

2018.5.25 Style and docstring fixes.

2016.2.25 Fixed some ionization energies.

References

(1) http://physics.nist.gov/PhysRefData/Compositions/

(2) http://physics.nist.gov/PhysRefData/IonEnergy/tblNew.html

(3) http://en.wikipedia.org/wiki/%(element.name)s

(4) http://www.miranda.org/~jkominek/elements/elements.db

Examples

```
>>> from elements import ELEMENTS
>>> len(ELEMENTS)
109
>>> str(ELEMENTS[109])
'Meitnerium'
>>> ele = ELEMENTS['C']
>>> ele.number, ele.symbol, ele.name, ele.eleconfig
(6, 'C', 'Carbon', '[He] 2s2 2p2')
>>> from pprint import pprint
>>> pprint(ele.eleconfig_dict)
{(1, 's'): 2, (2, 'p'): 2, (2, 's'): 2}
>>> sum(ele.mass for ele in ELEMENTS)
14659.1115599
>>> for ele in ELEMENTS:
...     ele.validate()
...     ele = eval(repr(ele))
```

Functions:

ELEMENTS	Ordered dict of Elements with lookup by number, symbol, and name.
Element(number, symbol, name, **kwargs)	Chemical element.
Isotope([mass, abundance, massnumber])	Isotope massnumber, relative atomic mass, and abundance.

class xfluo.elements.**Element**(*number*, *symbol*, *name*, ***kwargs*)

Bases: object

Chemical element.

number
Atomic number

Type int

symbol
Chemical symbol

Type str of length 1 or 2

name
Name in english

Type str

group
Group in periodic table

Type int

period
Period in periodic table

Type int

block
Block in periodic table

Type int

series
Index to chemical series

Type int

protons
Number of protons

Type int

neutrons
Number of neutrons in the most abundant naturally occurring stable isotope

Type int

nominalmass
Mass number of the most abundant naturally occurring stable isotope

Type int

electrons
Number of electrons

Type int

mass
Relative atomic mass. Ratio of the average mass of atoms of the element to 1/12 of the mass of an atom of 12C

Type float

exactmass
Relative atomic mass calculated from the isotopic composition

Type float

eleneg
Electronegativity (Pauling scale)

Type float

covrad
Covalent radius in Angstrom

Type float

atmrad
Atomic radius in Angstrom

vdwrad
Van der Waals radius in Angstrom

Type float

tboil
Boiling temperature in K

Type float

tmelt
Melting temperature in K

Type float

density
Density at 295K in g/cm3 respectively g/L

Type float

oxistates
Oxidation states

Type str

eleaffin
Electron affinity in eV

Type float

eleconfig
Ground state electron configuration

Type str

eleconfig_dict
Ground state electron configuration (shell, subshell): electrons

Type dict

eleshells
Number of electrons per shell

Type int

ionenergy
Ionization energies in eV

Type tuple

isotopes
Isotopic composition. keys: isotope mass number values: Isotope(relative atomic mass, abundance)

Type dict

description
Lazy object attribute whose value is computed on first access.

eleconfig_dict
Lazy object attribute whose value is computed on first access.

eleshells
Lazy object attribute whose value is computed on first access.

exactmass
Lazy object attribute whose value is computed on first access.

neutrons
Lazy object attribute whose value is computed on first access.

nominalmass
Lazy object attribute whose value is computed on first access.

validate()
Check consistency of data. Raise Error on failure.

class xfluo.elements.**Isotope**(*mass=0.0*, *abundance=1.0*, *massnumber=0*)
Bases: object

Isotope massnumber, relative atomic mass, and abundance.

abundance

mass

massnumber

3.4.3 xfluo.file_io.reader

Module for importing raw data files.

Functions:

find_elements(channel_names)	Extract a sorted element list from a channel list
read_theta(path_files, theta_index, hdf_tag)	Reads hdf file and returns theta
read_projection(fname, element, hdf_tag, ...)	Reads a projection for a given element from a single xrf hdf file

Continued on next page

Table 3 – continued from previous page

read_channel_names(fname, hdf_tag, channel_tag)	Read the channel names
read_mic_xrf(path_files, element_index, ...)	Converts hdf files to numpy arrays for plotting and manipulation

xfluo.file_io.reader.**find_elements**(*channel_names*)
Extract a sorted element list from a channel list

Parameters **channel_names** (*list*) – List of channel names

Returns **elements** (*list*) – Sorted list of elements

xfluo.file_io.reader.**read_theta**(*path_files*, *theta_index*, *hdf_tag*)
Reads hdf file and returns theta

Parameters

- **path_files** (*list*) – List of path+filenames
- **theta_index** (*int*) – Index where theta is saved under in the hdf MAPS/extra_pvs_as_csv tag This is: 2-ID-E: 663; 2-ID-E (prior 2017): *657*; BNP: 8
- **hdf_tag** (*str*) – String defining the hdf5 data_tag name (ex. MAPS)

Returns **ndarray** (*ndarray*) – 1D array [theta]

xfluo.file_io.reader.**read_projection**(*fname*, *element*, *hdf_tag*, *roi_tag*, *channel_tag*)
Reads a projection for a given element from a single xrf hdf file

Parameters

- **fname** (*str*) – String defining the file name
- **element** (*str*) – String defining the element to select
- **hdf_tag** (*str*) – String defining the hdf5 data_tag name (ex. MAPS)
- **roi_tag** (*str*) – data tag for corresponding roi_tag (ex. XRF_roi)
- **channel_tag** (*str*) – String defining the hdf5 channel tag name (ex. channel_names)

Returns **ndarray** (*ndarray*) – projection

xfluo.file_io.reader.**read_channel_names**(*fname*, *hdf_tag*, *channel_tag*)
Read the channel names

Parameters

- **fname** (*str*) – String defining the file name
- **hdf_tag** (*str*) – String defining the hdf5 data_tag name (ex. MAPS)
- **channel_tag** (*str*) – String defining the hdf5 channel tag name (ex. channel_names)

Returns **channel_names** (*list*) – List of channel names

xfluo.file_io.reader.**read_mic_xrf**(*path_files*, *element_index*, *hdf_tag*, *roi_tag*, *channel_tag*)
Converts hdf files to numpy arrays for plotting and manipulation

Parameters

- **path_files** (*list*) – List of (path + filenames)
- **theta_index** (*int*) – Index where theta is saved under in the hdf MAPS/extra_pvs_as_csv tag This is: 2-ID-E: 663; 2-ID-E (prior 2017): *657*; BNP: 8

- **hdf_tag** (*str*) – String defining the hdf5 data_tag name (ex. MAPS)
- **roi_tag** (*str*) – data tag for corresponding roi_tag (ex. XRF_roi)
- **channel_tag** (*str*) – String defining the hdf5 channel tag name (ex. channel_names)

Returns **ndarray** (*ndarray*) – 4D array [elements, projection, y, x]

3.4.4 xfluo.models.file_table

Subclasses the h5py module for interacting with Data Exchange files.

Functions:

FileTableModel([parent])	Interact with Data Exchange files.
TableArrayItem(fname)	

class xfluo.models.file_table.**TableArrayItem**(*fname*)

class xfluo.models.file_table.**FileTableModel**(*parent=None*, **args*)

Bases: sphinx.ext.autodoc.importer._MockObject

Interact with Data Exchange files.

data(*self*, *index*, *role*)

Helper function for ....

loadDirectory(*self*, *directoryName*, *ext='*.h5*'*)

This method is used to ....

columnCount(*parent*)

data(*index*, *role*)

flags(*index*)

getFirstCheckedFilePath()

loadDirectory(*directoryName*, *ext='*.h5*'*)

loadThetas(*img_tag*)

loadThetasLegacy(*thetaPV*)

rowCount(*parent*)

setAllChecked(*value*)

setChecked(*rows*, *value*)

setData(*index*, *value*, *role=<sphinx.ext.autodoc.importer._MockObject object>*)

sort(*col*, *order*)

Sort table by given column number.

3.4.5 xfluo.models.element_table

Functions:

ElementTableModel([parent])

class xfluo.models.element_table.**ElementTableModel**(*parent=None*, **args*)

Bases: sphinx.ext.autodoc.importer._MockObject

columnCount(*parent*)

data(*index*, *role*)

flags(*index*)

loadElementNames(*filePath*, *image_tag*, *element_tag*)

rowCount(*parent*)

setAllChecked(*value*)

setChecked(*rows*, *value*)

setData(*index*, *value*, *role=<sphinx.ext.autodoc.importer._MockObject object>*)

3.5 Examples

Here we describe what the examples are doing. You can cite with [B1].

3.5.1 HDF reader

This section contains an example of how to read a projection from an hdf file for a given element.

Download file: example_01.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ######################################################################
# Copyright (c) 2018, UChicago Argonne, LLC. All rights reserved.      #
#                                                                      #
# Copyright 2018. UChicago Argonne, LLC. This software was produced    #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National   #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the    #
# U.S. Department of Energy. The U.S. Government has rights to use,    #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR    #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR     #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is     #
# modified to produce derivative works, such modified software should     #
# be clearly marked, so as not to confuse it with the version available   #
# from ANL.                                                            #
#                                                                      #
# Additionally, redistribution and use in source and binary forms, with   #
# or without modification, are permitted provided that the following      #
# conditions are met:                                                  #
#                                                                      #
#     * Redistributions of source code must retain the above copyright    #
#       notice, this list of conditions and the following disclaimer.     #
#                                                                      #
#     * Redistributions in binary form must reproduce the above copyright #
```

(continues on next page)

(continued from previous page)

```
#       notice, this list of conditions and the following disclaimer in   #
#       the documentation and/or other materials provided with the        #
#       distribution.                                                     #
#                                                                         #
#     * Neither the name of UChicago Argonne, LLC, Argonne National       #
#       Laboratory, ANL, the U.S. Government, nor the names of its        #
#       contributors may be used to endorse or promote products derived   #
#       from this software without specific prior written permission.     #
#                                                                         #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS     #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT       #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       #
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago     #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,    #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER        #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT      #
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN       #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE         #
# POSSIBILITY OF SUCH DAMAGE.                                             #
# ########################################################################

"""
Example script
"""

from __future__ import print_function

import os
import sys
import argparse

import xfluo

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("fname", help="Directory containing multiple datasets or file␣
↪name of a single dataset: /data/ or /data/sample.h5")
    parser.add_argument("--start", nargs='?', type=float, default=0.0, help="Angle␣
↪first projection (default 0.0)")
    parser.add_argument("--end", nargs='?', type=float, default=180.0, help="Angle␣
↪last projection (default 180.0)")
    parser.add_argument("--element", nargs='?', type=str, default="Si", help="Select␣
↪the element to recontruct (default Si)")

    args = parser.parse_args()

    # Set path to the micro-CT data to reconstruct.
    fname = args.fname
    angle_start = float(args.start)
    angle_end = float(args.end)
    element = args.element

    if os.path.isfile(fname):

```

(continues on next page)

(continued from previous page)

```
        elements = xfluo.read_elements(fname)
        for i, e in enumerate(elements):
            print ('%d:  %s' % (i, e))

        proj, theta = xfluo.read_projection(fname, element, 663)
        print ("theta:", theta)
#         print (theta.shape)
        print ("proj:", proj)
        print (proj.shape)

    elif os.path.isdir(fname):
        # Add a trailing slash if missing
        top = os.path.join(fname, '')

        h5_file_list = list(filter(lambda x: x.endswith(('.h5', '.hdf')), os.
↪listdir(top)))

        for i, fname in enumerate(h5_file_list):
            proj, theta = xfluo.file_io.read_projection(top+fname, element, 663) #
↪#657
            print(i, theta, fname)

    else:
        print("Directory or File Name does not exist: ", fname)

if __name__ == "__main__":
    main(sys.argv[1:])
```

3.6 Credits

3.6.1 Citations

We kindly request that you cite the following article [A1] if you use project.

3.6.2 References

Bibliography

[A1] Gürsoy D, De Carlo F, Xiao X, and Jacobsen C. Tomopy: a framework for the analysis of synchrotron tomographic data. *Journal of Synchrotron Radiation*, 21(5):1188–1193, 2014.

[B1] De Carlo F, Gursoy D, Marone F, Rivers M, Parkinson YD, Khan F, Schwarz N, Vine DJ, Vogt S, Gleber SC, Narayanan S, Newville M, Lanzirotti T, Sun Y, Hong YP, and Jacobsen C. Scientific data exchange: a schema for hdf5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, 2014.

Python Module Index

x

Index

O

P

R

S

T

V

X