spec2nexus Documentation

Release 0.g2c26a11.dirty

Pete R. Jemian

Feb 24, 2022

Contents

Converts SPEC data files and scans into NeXus HDF5 files:

```
$ spec2nexus  path/to/file/specfile.dat
```

Writes `path/to/file/specfile.hdf5`

CHAPTER 1

Provides

- **spec2nexus** : command-line tool: Convert SPEC data files to NeXus HDF5
- **extractSpecScan** : command-line tool: Save columns from SPEC data file scan(s) to TSV files
- **spec** : library: python binding to read SPEC data files
- **eznx** : library: (Easy NeXus) supports writing NeXus HDF5 files using h5py
- **specplot** : command-line tool: plot a SPEC scan to an image file
- **specplot_gallery** : command-line tool: call **specplot** for all scans in a list of files, makes a web gallery

CHAPTER 2

Package Information

- **author**: Pete R. Jemian
- **email**: prjemian@gmail.com
- **copyright**: 2014-2020, Pete R. Jemian
- **license**: Creative Commons Attribution 4.0 International Public License (see LICENSE.txt file)
- **URL**: documentation: https://prjemian.github.io/spec2nexus/
- **git**: source: https://github.com/prjemian/spec2nexus
- **PyPI**: Distribution: https://pypi.python.org/pypi/spec2nexus/
- **OpenHub**: Compare open source software: https://www.openhub.net/p/spec2nexus
- **version**: 2021.1.11
- **release**: 0.g2c26a11.dirty
- **published**: Feb 24, 2022

2.1 Contents

2.1.1 spec2nexus

Converts SPEC data files and scans into NeXus HDF5 files.

How to use spec2nexus

Convert all scans in a SPEC data file:

```
$ spec2nexus  path/to/file/specfile.dat
```

Writes `path/to/file/specfile.hdf5` (Will not overwrite if the HDF5 exists, use the *-f* option to force overwrite).

show installed version

Verify the version of the installed spec2nexus:

```
$ spec2nexus  -v
2014.03.02
```

command-line options

```
user@host ~$ spec2nexus.py -h
usage: spec2nexus [-h] [-e HDF5_EXTENSION] [-f] [-v] [-s SCAN_LIST] [-t]
                  [--quiet | --verbose]
                  infile [infile ...]

spec2nexus: Convert SPEC data file into a NeXus HDF5 file.

positional arguments:
  infile                SPEC data file name(s)

optional arguments:
  -h, --help            show this help message and exit
  -e HDF5_EXTENSION, --hdf5-extension HDF5_EXTENSION
                        NeXus HDF5 output file extension, default = .hdf5
  -f, --force-overwrite
                        overwrite output file if it exists
  -v, --version         show program's version number and exit
  -s SCAN_LIST, --scan SCAN_LIST
                        specify which scans to save, such as: -s all or -s 1
                        or -s 1,2,3-5 (no spaces!), default = all
  --quiet                     suppress all program output (except errors), do not
                        use with --verbose option
  --verbose                   print more program output, do not use with --quiet
                        option
```

Note: Where's the source code to spec2nexus?

In the source code, the *spec2nexus* program is started from file **nexus.py** (in the `spec2nexus.nexus.main()` method, for those who look at the source code):

```
$ python nexus.py specfile.dat
```

You're not really going to call that from the source directory, are you? It will work, *if* you have put that source directory on your PYTHONPATH.

source code documentation

2.1.2 extractSpecScan

Command line tool to extract scan data from a SPEC data file.

How to use extractSpecScan

Extract one scan from a SPEC data file:

```
user@host ~$ extractSpecScan data/APS_spec_data.dat -s 1 -c mr USAXS_PD I0 seconds
```

the usage message:

```
user@host ~$ extractSpecScan
usage: extractSpecScan [-h] [-v] [--nolabels] -s SCAN [SCAN ...] -c COLUMN
                       [COLUMN ...] [-G] [-P] [-Q] [-V] [--quiet | --verbose]
                       spec_file
```

the version number:

```
user@host ~$ extractSpecScan -v
2017.0201.0
```

the help message:

```
user@host ~$ extractSpecScan -h
usage: extractSpecScan [-h] [-v] [--nolabels] -s SCAN [SCAN ...] -c COLUMN
                       [COLUMN ...] [-G] [-P] [-Q] [-V] [--quiet | --verbose]
                       spec_file

Save columns from SPEC data file scan(s) to TSV files URL:
https://prjemian.github.io/spec2nexus//extractSpecScan.html v2016.1025.0

positional arguments:
  spec_file             SPEC data file name(s)

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         print version number and exit
  --nolabels            do not write column labels to output file (default:
                        write labels)
  -s SCAN [SCAN ...], --scan SCAN [SCAN ...]
                        scan number(s) to be extracted (must specify at least
                        one)
  -c COLUMN [COLUMN ...], --column COLUMN [COLUMN ...]
                        column label(s) to be extracted (must specify at least
                        one)
  -G                    report scan Geometry (#G) header information
  -P                    report scan Positioners (#O & #P) header information
  -Q                    report scan Q (#Q) header information
  -V                    report scan (UNICAT-style #H & #V) header information
  --quiet               suppress all program output (except errors), do not
                        use with --verbose option
  --verbose             print more program output, do not use with --quiet
                        option
```

Example

Extract four columns (mr, USAXS_PD, I0, seconds) from two scans (1, 6) in a SPEC data file:

```
$ extractSpecScan data/APS_spec_data.dat -s 1 6 -c mr USAXS_PD I0 seconds

program: /path/to/extractSpecScan.py
read: data/APS_spec_data.dat
wrote: data/APS_spec_data_1.dat
wrote: data/APS_spec_data_6.dat
```

Here's the contents of *data/APS_spec_data_6.dat*:

```
# mr  USAXS_PD I0 seconds
15.61017 9.0   243.0 0.3
15.61 13.0  325.0 0.3
15.60984 19.0  460.0 0.3
15.60967 30.0  609.0 0.3
15.6095  54.0  883.0 0.3
15.60934 161.0 1780.0   0.3
15.60917 499.0 3649.0   0.3
15.609   1257.0   6588.0   0.3
15.60884 2832.0   10245.0  0.3
15.60867 7294.0   13118.0  0.3
15.6085  139191.0 16527.0  0.3
15.60834 299989.0 17893.0  0.3
15.60817 299989.0 18276.0  0.3
15.608   299989.0 18240.0  0.3
15.60784 299989.0 18266.0  0.3
15.60767 299989.0 18616.0  0.3
15.6075  299989.0 19033.0  0.3
15.60734 299989.0 19036.0  0.3
15.60717 299988.0 18587.0  0.3
15.607   299989.0 17471.0  0.3
15.60684 123003.0 14814.0  0.3
15.60667 11060.0  11861.0  0.3
15.6065  2217.0   8131.0   0.3
15.60634 637.0 4269.0   0.3
15.60617 254.0 2632.0   0.3
15.606   132.0 1927.0   0.3
15.60584 79.0  1406.0   0.3
15.60567 58.0  1075.0   0.3
15.6055  32.0  695.0 0.3
15.60534 17.0  374.0 0.3
15.60517 10.0  245.0 0.3
```

source code documentation

2.1.3 specplot

Read a SPEC data file and plot a thumbnail image.

This code can be called as a standalone program or it can be imported into another program and called as a subroutine, as shown in the *specplot_gallery* program.

The standard representation of a SPEC scan is a line plot of the last data column *versus* the first data column. Any

SPEC macro which name ends with *scan* ([1]) will be plotted as a line plot.

A special case SPEC scan macro is the *hklscan* where one of the three reciprocal space axes is scanned while the other two remain constant. A special handler (*SPEC's hklscan macro*) is provided to pick properly the scanned axis (not always the first column) for representation as a line plot.

Some SPEC macros scan two positioners over a grid to collect a 2-D image one pixel at a time. These scans are represented as color-mapped images where the first two columns are the vertical and horizontal axes and the image is color-mapped to intensity. Any SPEC macro which name ends with *mesh* will be plotted as an image plot.

Different handling can be customized for scan macros, as described in *How to write a custom scan handling for specplot*.

How to use *specplot*

Plot a scan from one of the sample data files supplied with *spec2nexus*:

```
user@host ~$ specplot src/spec2nexus/data/APS_spec_data.dat 2 specplot.png
```

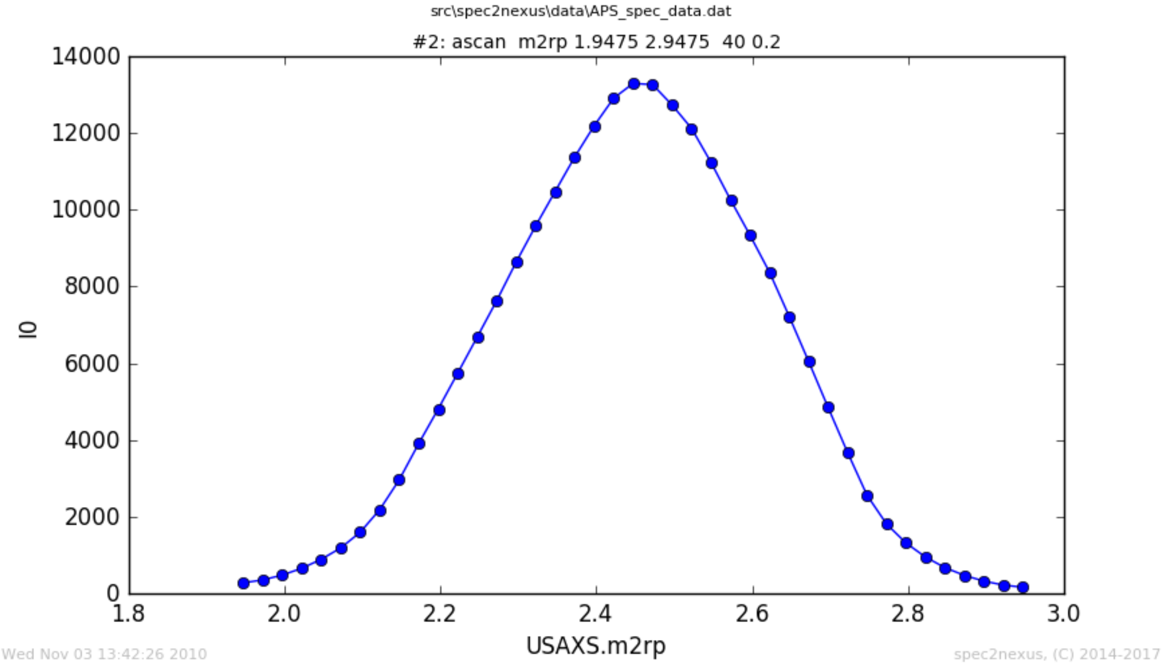


Fig. 1: Plot of scan #2 from example data file *APS_spec_data.dat*.

Usage

```
user@host ~$ specplot
usage: specplot.py [-h] specFile scan_number plotFile
```

[1] *scan*: any scan where the last four letters converted to lower case match *scan*, such as *ascan*, *a2scan*, *Escan*, *tscan*, *uascan*, *FlyScan*, *unusual_custom_user_scan*, . . .

Help

```
user@host ~$ specplot -h
usage: specplot.py [-h] specFile scan_number plotFile

read a SPEC data file and plot scan n

positional arguments:
  specFile     SPEC data file name
  scan_number  scan number in SPEC file
  plotFile     output plot file name

optional arguments:
  -h, --help   show this help message and exit
```

source code documentation

2.1.4 specplot_gallery

Read a list of SPEC data files (or directory(s) containing SPEC data files) and plot images of all scans. *specplot_gallery* will store these images in subdirectories of the given base directory (default: current directory) based on this structure:

```
{base directory}
   /{year}
      /{month}
         /{spec file name}
             /index.html
              s00001.png
              s00002.png
```

The year and month are taken from the SPEC data file when the data were collected. The plot names include the scan numbers padded with leading zeroes to five places (so the file names sort numerically).

The results will be shown as a WWW page (*index.html*) of thumbnail images *and* a separate list of any scans that could not generate plots. A reason will accompany these scans, as shown in the example.

How to use *specplot_gallery*: command line

Here is an example:

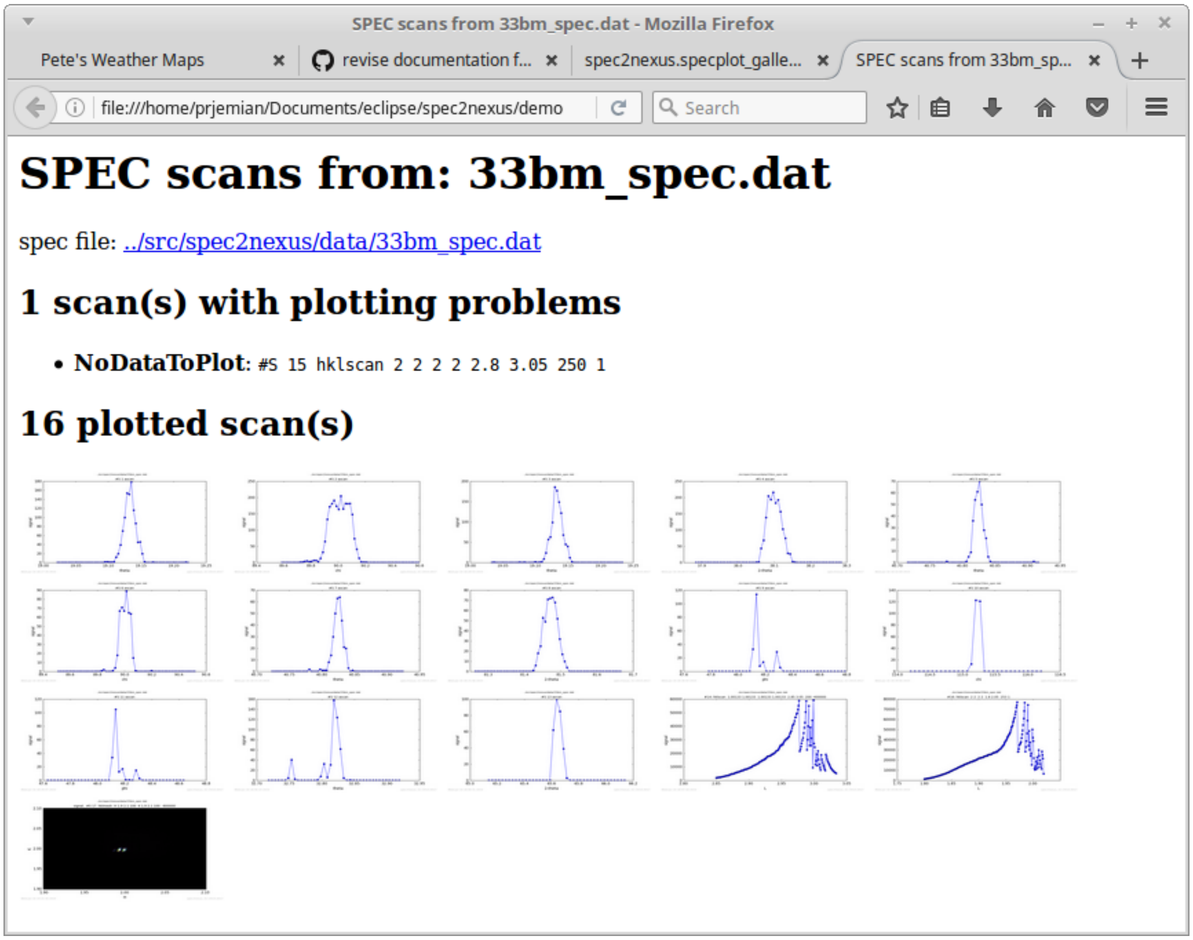
```
user@host ~$ specplot_gallery -d ./__demo__ ../src/spec2nexus/data/33bm_spec.dat
```

Note that one of the scans could not be plotted. Looking at the data file, it shows there is *no data to plot* (this particular scan was aborted before any data was collected):

```
#C Wed Jun 16 19:00:10 2010.  Scan aborted after 0 points.
```

The last scan shown is from a *hklmesh* (2-D) scan. It is mostly a constant background level, thus the large black area.

Each of the plots in the web page can be enlarged (by clicking on it).


SPEC scans from 33bm_spec.dat - Mozilla Firefox
Pete's Weather Maps
revise documentation f...
spec2nexus.specplot_galle...
SPEC scans from 33bm_sp...
file:///home/prjemian/Documents/eclipse/spec2nexus/demo
Search

SPEC scans from: 33bm_spec.dat

spec file: ../src/spec2nexus/data/33bm_spec.dat

1 scan(s) with plotting problems

- **NoDataToPlot**: `#S 15 hklscan 2 2 2 2 2.8 3.05 250 1`

16 plotted scan(s)

Fig. 2: Example of *specplot_gallery* showing scans from test file *33bm_spec.dat*.

How to use *specplot_gallery*: periodic background task (cron)

This script could be called from a Linux background task scheduler (*cron*) entry. To add the entry, type the *crontab -e* command which opens the task list in a screen editor and add lines such as these to the file:

```
# every five minutes (generates no output from outer script)
0-59/5 * * * *  /path/to/specplot_gallery.py -d /web/page/dir /spec/data/file/dirs
```

If the *specplot_gallery* script is called too frequently and the list of plots to be generated is large enough, it is possible for more than one process to be running. In one extreme case, many processes were found running due to problems with the data files. To identify and stop all processes of this program, use this on the command line:

```
kill -9 `ps -ef | grep python | awk '/specplot_gallery.py/ {print $2}' -`
```

source code documentation

2.1.5 `spec2nexus.spec`

Library of classes to read the contents of a SPEC data file.

How to use `spec2nexus.spec`

`spec2nexus.spec` provides Python support to read the scans in a SPEC data file. (It does not provide a command-line interface.) Here is a quick example how to use `spec`:

```
from spec2nexus.spec import SpecDataFile

specfile = SpecDataFile('data/33id_spec.dat')
print 'SPEC file name:', specfile.specFile
print 'SPEC file time:', specfile.headers[0].date
print 'number of scans:', len(specfile.scans)

for scanNum, scan in specfile.scans.items():
    print scanNum, scan.scanCmd
```

For one example data file provided with `spec2nexus.spec`, the output starts with:

How to read one scan

Here is an example how to read one scan:

```
from spec2nexus.spec import SpecDataFile

specfile = SpecDataFile('data/33id_spec.dat')
specscan = specfile.getScan(5)
print specscan.scanNum
print specscan.scanCmd
```

which has this output:

```
5
ascan  del 84.3269 84.9269  30 1
```

The data columns are provided in a dictionary. Using the example above, the dictionary is `specscan.data` where the keys are the column labels (from the #L line) and the values are from each row. It is possible to make a default plot of the last column vs. the first column. Here's how to find that data:

```
x_label = specscan.L[0]          # first column from #L line
y_label = specscan.L[-1]         # last column from #L line
x_data = specscan.data[x_label]  # data for first column
y_data = specscan.data[y_label]  # data for last column
```

Get a list of the scans

The complete list of scan numbers from the data file is obtained (sorting is necessary since the list of dictionary keys is returned in a scrambled order):

```
all_scans = sorted(specfile.scans.keys())
```

SPEC data files

The SPEC data file format is described in the SPEC manual.[1] This manual is taken as a suggested starting point for most users. Data files with deviations from this standard are produced at some facilities.

Assumptions about data file structure

These assumptions are used to parse SPEC data files:

1. SPEC data files are text files organized by lines. The lines can be categorized as: **control lines**, **data lines**, and blank lines.

line type	description
control	contain a # character in the first column followed by a command word[2]
data	generally contain a row of numbers (the scan data)
special data	containing MCA data[3]

2. Lines in a SPEC data file start with a file name control line, then series of blocks. Each block may be either a file header block or a scan block. (Most SPEC files have only one header block. A new header block is created if the list of positioners is changed in SPEC without creating a new file. SPEC users are encouraged to *always* start a new data file after changing the list of positioners.) A block consists of a series of control, data, and blank lines.

   SPEC data files are composed of a sequence of a single file header block and zero or more scan blocks.[4]

3. A SPEC data file always begins with this control lines: #F, such as:

   ```
   #F samplecheck_7_17_03
   ```

4. A file header block begins with these control lines in order: #E #D #C, such as:

[1] SPEC manual: http://www.certif.com/spec_manual/user_1_4_1.html

[2] See *Example of Control Lines*

[3] See *Example of MCA data lines*

[4] It is very unusual to have more than one file header block in a SPEC data file.

```
#E 1058427452
#D Thu Jul 17 02:37:32 2003
#C psic  User = epix
```

5. A scan block begins with these command lines in order: #S #D, such as:

```
#S 78  ascan  del 84.6484 84.8484  20 1
#D Thu Jul 17 08:03:54 2003
```

Control lines (keys) defined by SPEC

Here is a list[5] of keys (command words) from the comments in the *file.mac* (SPEC v6) macro source file:

command word	description
#C	comment line
#D date	current date and time in UNIX format
#E num	the UNIX epoch (seconds from 00:00 GMT 1/1/70)
#F name	name by which file was created
#G1 …	geometry parameters from G[] array (geo mode, sector, etc)
#G2 …	geometry parameters from U[] array (lattice constants, orientation reflections)
#G3 …	geometry parameters from UB[] array (orientation matrix)
#G4 …	geometry parameters from Q[] array (lambda, frozen angles, cut points, etc)
#I num	a normalizing factor to apply to the data
#j% …	mnemonics of counter (% = 0,1,2,… with eight counters per row)
#J% …	names of counters (each separated by two spaces)
#L s1 …	labels for the data columns
#M num	data was counted to this many monitor counts
#N num [num2]	number of columns of data [num2 sets per row]
#o% …	mnemonics of motors (% = 0,1,2,… with eight motors per row)
#O% …	names of motors (each separated by two spaces)
#P% …	positions of motors corresponding to above #O/#o
#Q	a reciprocal space position (H K L)
#R	user-defined results from a scan
#S num	scan number
#T num	data was counted for this many seconds
#U	user defined
#X	a temperature
#@MCA fmt	this scan contains MCA data (array_dump() format, as in `"%16C"`)
#@CALIB a b c	coefficients for `x[i] = a + b * i + c * i * i` for MCA data
#@CHANN n f l r	MCA channel information (number_saved, first_saved, last_saved, reduction coef)
#@CTIME p l r	MCA count times (preset_time, elapsed_live_time, elapsed_real_time)
#@ROI n f l	MCA ROI channel information (ROI_name, first_chan, last_chan)

Example of Control Lines

The command word of a control line may have a number at the end, indicating it is part of a sequence, such as these control lines (see *Control lines (keys) defined by SPEC* for how to interpret):

[5] Compare with *Supplied spec plugin modules*

Example of MCA data lines

Lines with MCA array data begin with the **@A** command word. (If such a data line ends with a continuation character `\`, the next line is read as part of this line.)

This is an example of a 91-channel MCA data array with trivial (zero) values:

```
@A 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0
```

Several MCA spectra may be written to a scan. In this case, a number follows **@A** indicating which spectrum, such as in this example with four spectra:

```
     @A1 0 0 0 0 0 0 35 0 0 35
     @A2 0 0 0 0 0 0 0 35 0 35
     @A3 0 0 35 35 0 0 0 0 0 0
     @A4 0 0 0 0 0 35 35 0 35 0
```

Supported header keys (command words)

The SPEC data file keys recognized by `spec` are listed in *Supplied spec plugin modules*.

---

source code summary

classes

`spec2nexus.spec.SpecDataFile`
`spec2nexus.spec.SpecDataFileHeader`
`spec2nexus.spec.SpecDataFileScan`

methods

`strip_first_word`	return everything after the first space on the line from the spec data file
`spec2nexus.spec.is_spec_file`	

exceptions

`spec2nexus.spec.SpecDataFileNotFound`
`spec2nexus.spec.SpecDataFileCouldNotOpen`
`spec2nexus.spec.SpecDataFileNotFound`

Continued on next page

Table 3 – continued from previous page

`spec2nexus.spec.DuplicateSpecScanNumber`
`spec2nexus.spec.UnknownSpecFilePart`

dependencies

`os`	OS routines for NT or Posix depending on what system we're on.
`re`	Support for regular expressions (RE).
`sys`	This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.

internal structure of `spec2nexus.spec.SpecDataFileScan`

The internal variables of a Python class are called *attributes*. It may be convenient, for some, to think of them as *variables*.

scan attributes

parent *obj* - instance of `spec2nexus.spec.SpecDataFile`

scanNum *int* - SPEC scan number

scanCmd *str* - SPEC command line

raw *str* - text of scan, as reported in SPEC data file

scan attributes (variables) set after call to plugins

These attributes are only set *after* the scan's `interpret()` method is called. This method is called automatically when trying to read any of the following scan attributes:

comments *[str]* - list of all comments reported in this scan

data *{label,[number]}* - written by `spec2nexus.plugins.spec_common_spec2nexus.data_lines_postprocessing()`

data_lines *[str]* - raw data (and possibly MCA) lines with comment lines removed

date *str* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Date`

G *{key,[number]}* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Geometry`

I *float* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_NormalizingFactor`

header *obj* - instance of `spec2nexus.spec.SpecDataFileHeader`

L *[str]* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Labels`

M *str* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Monitor`

positioner *{key,number}* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Positioners.postprocess`

N *[int]* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_NumColumns`

P *[str]* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Positioners`

Q *[number]* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_HKL`

S *str* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_Scan`

T *str* - written by `spec2nexus.plugins.spec_common_spec2nexus.SPEC_CountTime`

V *{key,number|str}* - written by `spec2nexus.plugins.unicat_spec2nexus.UNICAT_MetadataValues`

column_first *str* - label of first (ordinate) data column

column_last *str* - label of last (abscissa) data column

internal use only - do not modify

These scan attributes are for internal use only and are not part of the public interface. Do not modify them or write code that depends on them.

postprocessors *{key,obj}* - dictionary of postprocessing methods

h5writers *{key,obj}* - dictionary of methods that write HDF5 structure

__lazy_interpret__ *bool* - Is *lazy* (on-demand) call to `interpret()` needed?

__interpreted__ *bool* - Has `interpret()` been called?

source code documentation

2.1.6 `spec2nexus.charts`

source code documentation

charting for spec2nexus

make_png(image, image_file[, axes, title, . . .])	read the image from the named HDF5 file and make a PNG file
xy_plot(x, y, plot_file[, title, subtitle, . . .])	with MatPlotLib, generate a plot of a scan (as if data from a scan in a SPEC file)

`spec2nexus.charts.`**`make_png`**(*image*, *image_file*, *axes=None*, *title='2-D data'*, *subtitle="*, *log_image=False*, *hsize=9*, *vsize=5*, *cmap='cubehelix'*, *xtitle=None*, *ytitle=None*, *timestamp_str=None*)

read the image from the named HDF5 file and make a PNG file

Test that the HDF5 file exists and that the path to the data exists in that file. Read the data from the named dataset, mask off some bad values, convert to log(image) and use Matplotlib to make the PNG file.

Parameters

- **`image`** (`obj`) – array of data to be rendered

- **image_file** (*str*) – name of image file to be written (path is optional)
- **log_image** (*bool*) – plot log(image)
- **hsize** (*int*) – horizontal size of the PNG image (default: 7)
- **hsize** – vertical size of the PNG image (default: 3)
- **cmap** (*str*) – colormap for the image (default: 'cubehelix'), 'jet' is another good one

Return str *image_file*

The HDF5 file could be a NeXus file, or some other layout.

spec2nexus.charts.**xy_plot**(*x*, *y*, *plot_file*, *title=None*, *subtitle=None*, *xtitle=None*, *ytitle=None*, *xlog=False*, *ylog=False*, *hsize=9*, *vsize=5*, *timestamp_str=None*)

with MatPlotLib, generate a plot of a scan (as if data from a scan in a SPEC file)

Parameters

- **x** (*[float]*) – horizontal axis data
- **y** (*[float]*) – vertical axis data
- **plot_file** (*str*) – file name to write plot image
- **xtitle** (*str*) – horizontal axis label (default: not shown)
- **ytitle** (*str*) – vertical axis label (default: not shown)
- **title** (*str*) – title for plot (default: date time)
- **subtitle** (*str*) – subtitle for plot (default: not shown)
- **xlog** (*bool*) – should X axis be log (default: False=linear)
- **ylog** (*bool*) – should Y axis be log (default: False=linear)
- **timestamp_str** (*str*) – date to use on plot (default: now)

Tip: when using this module as a background task . . .

MatPlotLib has several interfaces for plotting. Since this module runs as part of a background job generating lots of plots, MatPlotLib's standard `plt` code is not the right model. It warns after 20 plots and will eventually run out of memory.

Here's the fix used in this module: http://stackoverflow.com/questions/16334588/create-a-figure-that-is-reference-counted/16337909#16337909

2.1.7 How to write a custom scan handling for *specplot*

Sometimes, it will be obvious that a certain scan macro never generates any plot images, or that the default handling creates a plot that is a poor representation of the data, such as the *hklscan* where only one of the the axes *hkl* is scanned. To pick the scanned axis for plotting, it is necessary to prepare custom handling and replace the default handling.

Overview

It is possible to add in additional handling by writing a Python module. This module creates a subclass of the standard handling, such as `LinePlotter`, `MeshPlotter`, or their superclass `ImageMaker`. The support is added to the macro selection class `Selector` with code such as in the brief example described below: *Change the plot title text in ascan macros*:

```
selector = spec2nexus.specplot.Selector()
selector.add('ascan', Custom_Ascan)
spec2nexus.specplot_gallery.main()
```

Data Model

The data to be plotted is kept in an appropriate subclass of `PlotDataStructure` in attributes show in the next table. The data model is an adaptation of the NeXus *NXdata* base class.[1]

attribute	description
self.signal	name of the dependent data (*y* axis or image) to be plotted
self.axes	list of names of the independent axes[2]
self.data	dictionary with the data, indexed by name

Steps

In all cases, custom handling of a specific SPEC macro name is provided by creating a subclass of `ImageMaker` and defining one or more of its methods. In the simplest case, certain settings may be changed by calling `spec2nexus.specplot.ImageMaker.configure()` with the custom values. Examples of further customization are provided below, such as when the data to be plotted is stored outside of the SPEC data file. This is common for images from area detectors.

It may also be necessary to create a subclass of `PlotDataStructure` to gather the data to be plotted or override the default `spec2nexus.specplot.ImageMaker.plottable()` method. An example of this is shown with the `MeshPlotter` and associated `MeshStructure` classes.

Examples

A few exmaples of custom macro handling are provided, some simple, some complex. In each example, decisions have been made about where to provide the desired features.

Change the plot title text in *ascan* macros

The SPEC *ascan* macro is a workhorse and records the scan of a positioner and the measurement of data in a counter. Since this macro name ends with "scan", the default selection in *specplot* images this data using the `LinePlotter` class. Here is a plot of the default handling of data from the *ascan* macro:

We will show how to change the plot title as a means to illustrate how to customize the handling for a scan macro.

We write `Custom_Ascan` which is a subclass of `LinePlotter`. The `get_plot_data` method is written (overrides the default method) to gain access to the place where we can introduce the change. The change is made by the call to the `configure` method (defined in the superclass). Here's the code:

ascan.py example

[1] NeXus *NXdata* base class: http://download.nexusformat.org/doc/html/classes/base_classes/NXdata.html

[2] The number of names provided in *self.axes* is equal to the *rank* of the *signal* data (*self.data[self.signal]*). For 1-D data, *self.axes* has one name and the *signal* data is one-dimensional. For 2-D data, *self.axes* has two names and the *signal* data is two-dimensional.

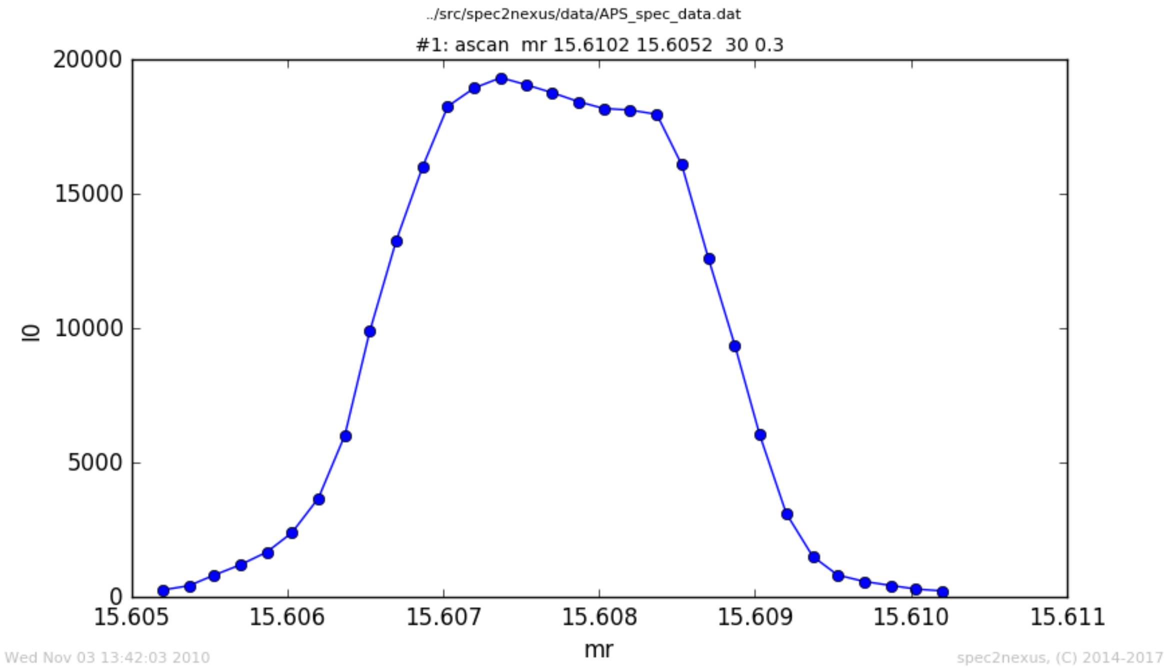


Fig. 3: Standard plot of data from *ascan* macro

```python
#!/usr/bin/env python

'''
Plot all scans that used the SPEC `ascan` macro, showing only the scan number (not full scan command)

This is a simple example of how to customize the scan macro handling.
There are many more ways to add complexity.
'''

import spec2nexus.specplot
import spec2nexus.specplot_gallery


class Custom_Ascan(spec2nexus.specplot.LinePlotter):
    '''simple customization'''

    def retrieve_plot_data(self):
        '''substitute with the data&time the plot was created'''
        import datetime
        spec2nexus.specplot.LinePlotter.retrieve_plot_data(self)
        self.set_plot_subtitle(str(datetime.datetime.now()))


def main():
    selector = spec2nexus.specplot.Selector()
    selector.add('ascan', Custom_Ascan)
    spec2nexus.specplot_gallery.main()


```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main()

# -----------------------------------------------------------------------------
# :author:    Pete R. Jemian
# :email:     prjemian@gmail.com
# :copyright: (c) 2014-2022, Pete R. Jemian
#
# Distributed under the terms of the Creative Commons Attribution 4.0 International␣
↪Public License.
#
# The full license is in the file LICENSE.txt, distributed with this software.
# -----------------------------------------------------------------------------
```

See the changed title:

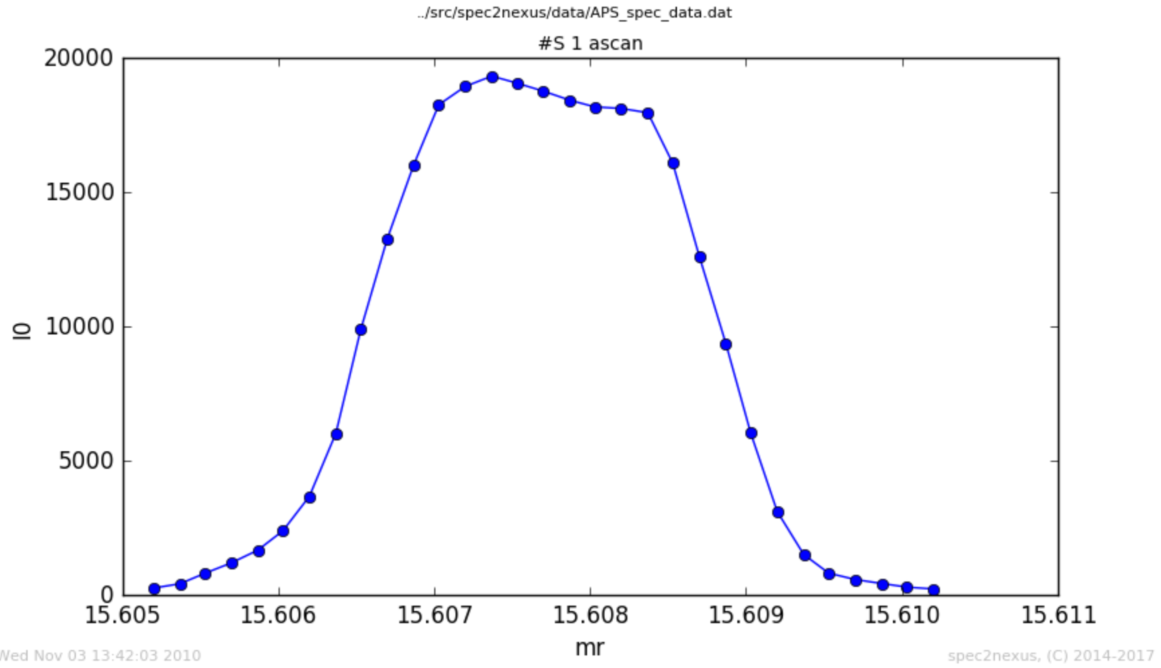


Fig. 4: Customized plot of data from *ascan* macro

Make the *y*-axis log scale

A very simple customization can make the Y axis to be logarithmic scale. (This customization is planned for an added feature[3] in a future relase of the *spec2nexus* package.) We present two examples.

modify handling of *a2scan*

One user wants all the *a2scan* images to be plotted with a logarithmic scale on the Y axis. Here's the code:

[3] specplot: add option for default log(signal)

custom_a2scan_gallery.py example

```python
#!/usr/bin/env python

'''
Customization for specplot_gallery: plot a2scan with log(y) axis

This program changes the plotting for all scans that used the *a2scan* SPEC macro.
The Y axis of these plots will be plotted as logarithmic if all the data values are
greater than zero.  Otherwise, the Y axis scale will be linear.
'''

import spec2nexus.specplot
import spec2nexus.specplot_gallery

class Custom_a2scan_Plotter(spec2nexus.specplot.LinePlotter):
    '''plot `a2scan` y axis as log if possible'''

    def retrieve_plot_data(self):
        '''plot the vertical axis on log scale'''
        spec2nexus.specplot.LinePlotter.retrieve_plot_data(self)

        choose_log_scale = False

        if self.signal in self.data:    # log(y) if all data positive
            choose_log_scale = min(self.data[self.signal]) > 0

        self.set_y_log(choose_log_scale)


def main():
    selector = spec2nexus.specplot.Selector()
    selector.add('a2scan', Custom_a2scan_Plotter)
    spec2nexus.specplot_gallery.main()


if __name__ == '__main__':
    # debugging_setup()
    main()

'''
Instructions:

Save this file in a directory you can write and call it from your cron tasks.

Note that in cron entries, you cannot rely on shell environment variables to
be defined.  Best to spell things out completely.  For example, if your $HOME
directory is `/home/user` and you have these directories:

* `/home/user/bin`: various custom executables you use
* `/home/user/www/specplots`: a directory you access with a web browser for your plots
* `/home/user/spec/data`: a directory with your SPEC data files

then save this file to `/home/user/bin/custom_a2scan_gallery.py` and make it executable
(using `chmod +x ./home/user/bin/custom_a2scan_gallery.py`).

```

(continues on next page)

(continued from previous page)

```
Edit your list of cron tasks using `crontab -e` and add this (possibly
replacing a call to `specplot_gallery` with this call `custom_a2scan_gallery.py`)::

    # every five minutes (generates no output from outer script)
    0-59/5 * * * *  /home/user/bin/custom_a2scan_gallery.py -d /home/user/www/
↪specplots /home/user/spec/data 2>&1 >> /home/user/www/specplots/log_cron.txt

Any output from this periodic task will be recorded in the file
`/home/user/www/specplots/log_cron.txt`.  This file can be reviewed
for diagnostics or troubleshooting.
'''
```

custom *uascan*

The APS USAXS instrument uses a custom scan macro called *uascan* for routine step scans. Since this macro name ends with “scan”, the default selection in *specplot* images this data using the `LinePlotter` class. Here is a plot of the default handling of data from the *uascan* macro:

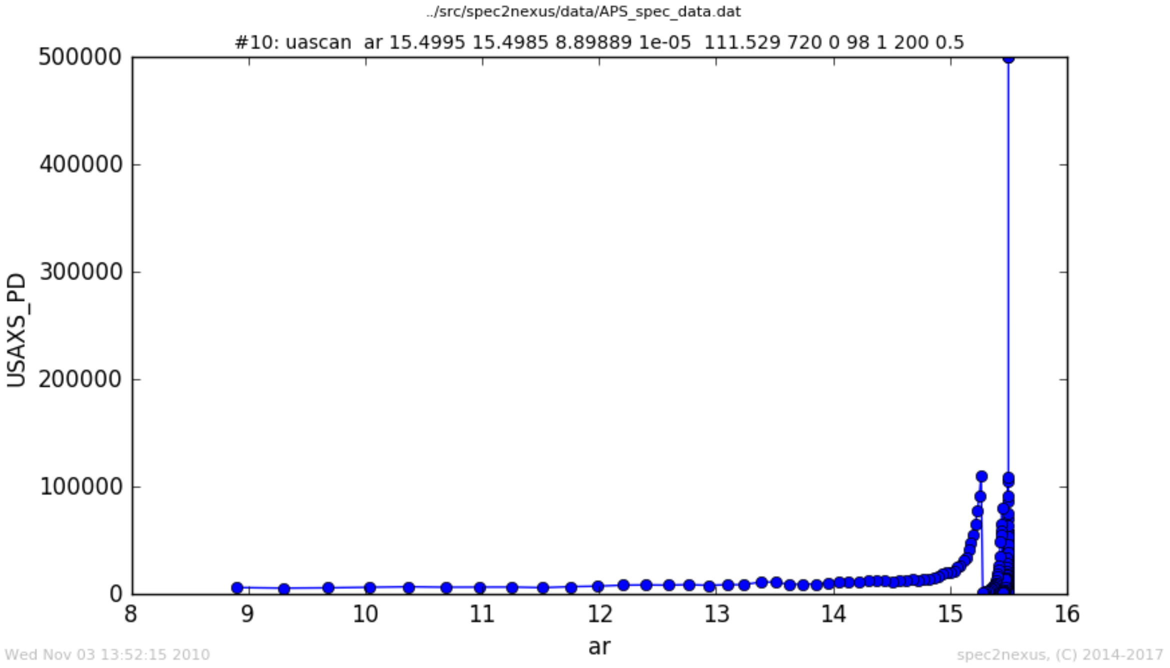


Fig. 5: USAXS *uascan*, handled as `LinePlotter`

The can be changed by making the *y* axis log scale. To do this, a custom version of `LinePlotter` is created as `Custom_Ascan`. The `get_plot_data` method is written (overrides the default method) to make the *y* axis log-scale by calling the `configure` method (defined in the superclass). Here’s the code:

usaxs_uascan.py example

```
#!/usr/bin/env python

```

(continues on next page)

(continued from previous page)

```
'''
Plot data from the USAXS uascan macro

.. autosummary::

    ~UAscan_Plotter

'''

import spec2nexus.specplot
import spec2nexus.specplot_gallery


class UAscan_Plotter(spec2nexus.specplot.LinePlotter):
    '''simple customize of `uascan` handling'''

    def retrieve_plot_data(self):
        '''plot the vertical axis on log scale'''
        spec2nexus.specplot.LinePlotter.retrieve_plot_data(self)

        if self.signal in self.data:
            if min(self.data[self.signal]) <= 0:
                # TODO: remove any data where Y <= 0 (can't plot on log scale)
                msg = 'cannot plot Y<0: ' + str(self.scan)
                raise spec2nexus.specplot.NotPlottable(msg)

        # in the uascan, a name for the sample is given in `self.scan.comments[0]`
        self.set_y_log(True)
        self.set_plot_subtitle(
            '#%s uascan: %s' % (str(self.scan.scanNum), self.scan.comments[0]))


def debugging_setup():
    import os, sys
    import shutil
    import ascan
    selector = spec2nexus.specplot.Selector()
    selector.add('ascan', ascan.Custom_Ascan)    # just for the demo
    path = '__usaxs__'
    shutil.rmtree(path, ignore_errors=True)
    os.mkdir(path)
    sys.argv.append('-d')
    sys.argv.append(path)
    sys.argv.append(os.path.join('..', 'src', 'spec2nexus', 'data', 'APS_spec_data.dat
↪'))


def main():
    selector = spec2nexus.specplot.Selector()
    selector.add('uascan', UAscan_Plotter)
    spec2nexus.specplot_gallery.main()


if __name__ == '__main__':
    # debugging_setup()
    main()

```

(continues on next page)

(continued from previous page)

```
# -----------------------------------------------------------------------------
# :author:    Pete R. Jemian
# :email:     prjemian@gmail.com
# :copyright: (c) 2014-2022, Pete R. Jemian
#
# Distributed under the terms of the Creative Commons Attribution 4.0 International Public License.
#
# The full license is in the file LICENSE.txt, distributed with this software.
# -----------------------------------------------------------------------------
```

Note that in the *uascan*, a name for the sample provided by the user is given in *self.scan.comments[0]*. The plot title is changed to include this and the scan number. The customized plot has a logarithmic *y* axis:

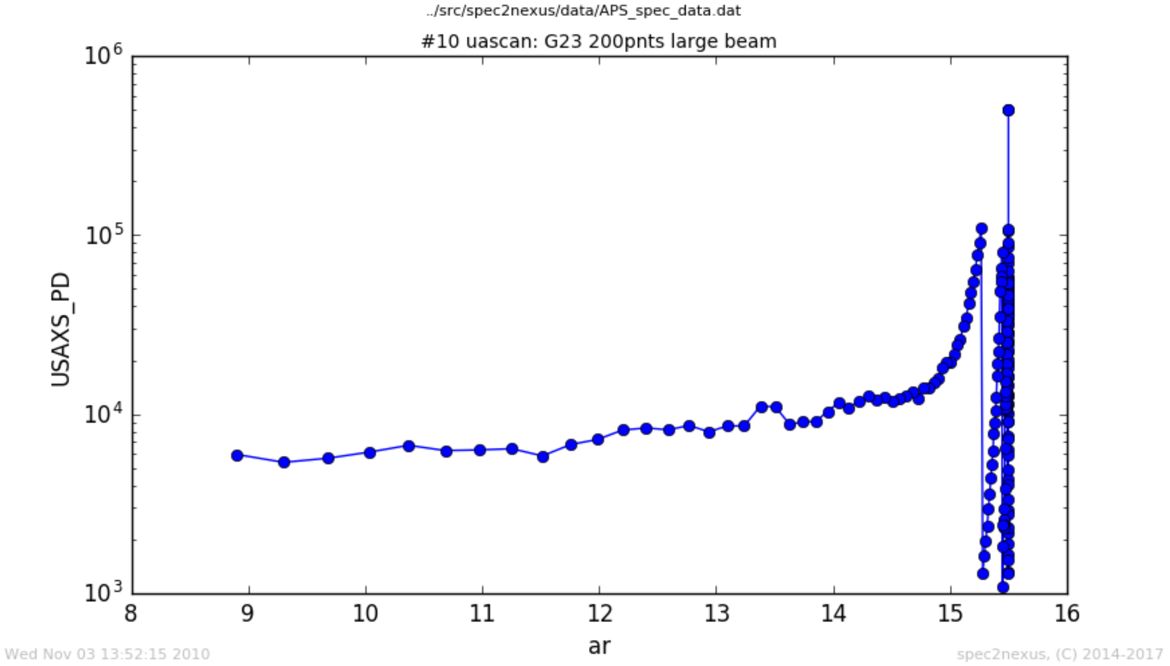


Fig. 6: USAXS *uascan*, with logarithmic *y* axis

The most informative view of this data is when the raw data are reduced to $I(Q)$ and viewed on a log-log plot, but that process is beyond this simple example. See the example *Get xy data from HDF5 file* below.

SPEC's *hklscan* macro

The SPEC *hklscan* macro appears in a SPEC data file due to either a *hscan*, *kscan*, or *lscan*. In each of these one of the *hkl* vectors is scanned while the other two remain constant.

The normal handling of the *ascan* macro plots the last data column against the first. This works for data collected with the *hscan*. For *kscan* or *lscan* macros, the *h* axis is still plotted by default since it is in the first column.

To display the scanned axis, it is necessary to examine the data in a custom subclass of `LinePlotter`. The `HKLScanPlotter` subclass, provided with *specplot*, defines the `get_plot_data()` method determines the scanned axis, setting it by name:

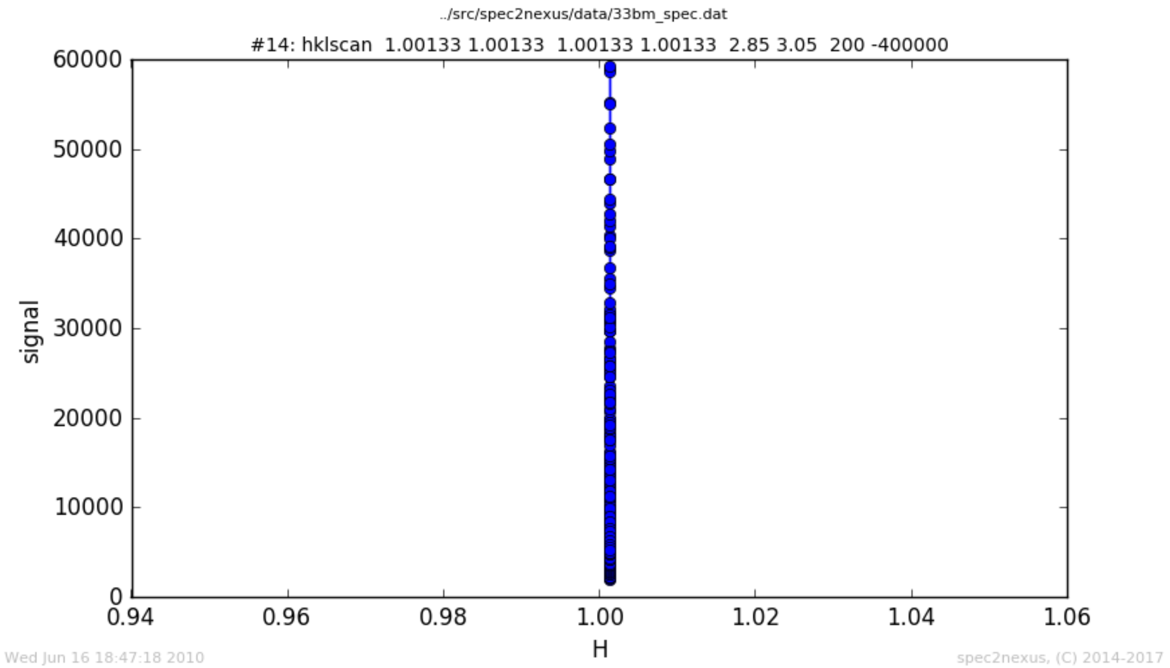


Fig. 7: SPEC *hklscan* (*lscan*, in this case), plotted against the (default) first axis *H*

```
plot.axes = [axis,]
self.scan.column_first = axis
```

Then, the standard plot handling used by *LinePlotter* uses this information to make the plot.

Get *xy* data from HDF5 file

One example of complexity is when SPEC has been used to direct data collection but the data is not stored in the SPEC data file. The SPEC data file scan must provide some indication about where the collected scan data has been stored.

The USAXS instrument at APS has a *FlyScan* macro that commands the instrument to collect data continuously over the desired Q range. The data is written to a NeXus HDF5 data file. Later, a data reduction process converts the arrays of raw data to one-dimensional $I(Q)$ profiles. The best representation of this reduced data is on a log-log plot to reveal the many decades of both I and Q covered by the measurement.

With the default handling by `LinePlotter`, no plot can be generated since the dfata is given in a separate HDF5 file. That file is read with the custom handling of the *usaxs_flyscan.py* demo:

usaxs_flyscan.py example

```
#!/usr/bin/env python

'''
Plot data from the USAXS FlyScan macro

.. autosummary::

```

(continues on next page)

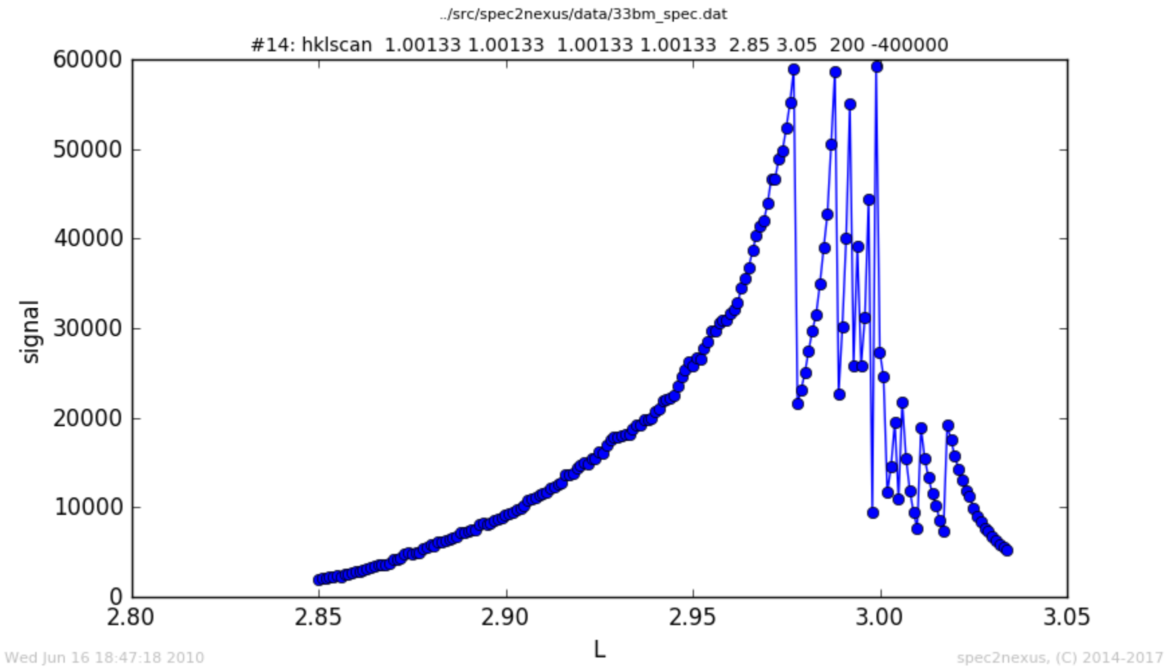


Fig. 8: SPEC *hklscan* (*lscan*), plotted against *L*

(continued from previous page)

```
    ~read_reduced_fly_scan_file
    ~retrieve_flyScanData
    ~USAXS_FlyScan_Structure
    ~USAXS_FlyScan_Plotter

'''

import h5py
import numpy
import os

import spec2nexus.specplot
import spec2nexus.specplot_gallery


# methods picked (& modified) from the USAXS livedata project
def read_reduced_fly_scan_file(hdf5_file_name):
    '''
    read any and all reduced data from the HDF5 file, return in a dictionary

    dictionary = {
      'full': dict(Q, R, R_max, ar, fwhm, centroid)
      '250':  dict(Q, R, dR)
      '5000': dict(Q, R, dR)
    }
    '''

    reduced = {}
    hdf = h5py.File(hdf5_file_name, 'r')
```

(continues on next page)

(continued from previous page)

```
    entry = hdf['/entry']
    for key in entry.keys():
        if key.startswith('flyScan_reduced_'):
            nxdata = entry[key]
            d = {}
            for dsname in ['Q', 'R']:
                if dsname in nxdata:
                    value = nxdata[dsname]
                    if value.size == 1:
                        d[dsname] = float(value[0])
                    else:
                        d[dsname] = numpy.array(value)
            reduced[key[len('flyScan_reduced_'):]] = d
    hdf.close()
    return reduced


# $URL: https://subversion.xray.aps.anl.gov/small_angle/USAXS/livedata/specplot.py $
REDUCED_FLY_SCAN_BINS   = 250       # the default
def retrieve_flyScanData(scan):
    '''retrieve reduced, rebinned data from USAXS Fly Scans'''
    path = os.path.dirname(scan.header.parent.fileName)
    key_string = 'FlyScan file name = '
    comment = scan.comments[2]
    index = comment.find(key_string) + len(key_string)
    hdf_file_name = comment[index:-1]
    abs_file = os.path.abspath(os.path.join(path, hdf_file_name))

    plotData = {}
    if os.path.exists(abs_file):
        reduced = read_reduced_fly_scan_file(abs_file)
        s_num_bins = str(REDUCED_FLY_SCAN_BINS)

        choice = reduced.get(s_num_bins) or reduced.get('full')

        if choice is not None:
            plotData = {axis: choice[axis] for axis in 'Q R'.split()}

    return plotData


class USAXS_FlyScan_Plotter(spec2nexus.specplot.LinePlotter):
    '''
    customize `FlyScan` handling, plot :math:`log(I)` *vs.* :math:`log(Q)`

    The USAXS FlyScan data is stored in a NeXus HDF5 file in a subdirectory
    below the SPEC data file.  This code uses existing code from the
    USAXS instrument to read that file.
    '''

    def retrieve_plot_data(self):
        '''retrieve reduced data from the FlyScan's HDF5 file'''
        # get the data from the HDF5 file
        fly_data = retrieve_flyScanData(self.scan)

        if len(fly_data) != 2:
            raise spec2nexus.specplot.NoDataToPlot(str(self.scan))
```

(continues on next page)

(continued from previous page)

```

        self.signal = 'R'
        self.axes = ['Q',]
        self.data = fly_data

        # customize the plot just a bit
        # sample name as given by the user?
        subtitle = '#' + str(self.scan.scanNum)
        subtitle += ' FlyScan: ' + self.scan.comments[0]
        self.set_plot_subtitle(subtitle)
        self.set_x_log(True)
        self.set_y_log(True)
        self.set_x_title(r'$|\vec{Q}|, 1/\AA$')
        self.set_y_title(r'USAXS $R(|\vec{Q}|)$, a.u.')

    def plottable(self):
        '''
        can this data be plotted as expected?
        '''
        if self.signal in self.data:
            signal = self.data[self.signal]
            if signal is not None and len(signal) > 0 and len(self.axes) == 1:
                if len(signal) == len(self.data[self.axes[0]]):
                    return True
        return False


def debugging_setup():
    import sys
    import shutil
    sys.path.insert(0, os.path.join('..', 'src'))
    path = '__usaxs__'
    shutil.rmtree(path, ignore_errors=True)
    os.mkdir(path)
    sys.argv.append('-d')
    sys.argv.append(path)
    sys.argv.append(os.path.join('..', 'src', 'spec2nexus', 'data', '02_03_setup.dat
↪'))


def main():
    selector = spec2nexus.specplot.Selector()
    selector.add('FlyScan', USAXS_FlyScan_Plotter)
    spec2nexus.specplot_gallery.main()


if __name__ == '__main__':
    # debugging_setup()
    main()

# -----------------------------------------------------------------------------
# :author:    Pete R. Jemian
# :email:     prjemian@gmail.com
# :copyright: (c) 2014-2022, Pete R. Jemian
#
# Distributed under the terms of the Creative Commons Attribution 4.0 International␣
↪Public License.
```

(continues on next page)

(continued from previous page)

```
#
# The full license is in the file LICENSE.txt, distributed with this software.
# -----------------------------------------------------------------------------
```

The data is then rendered in a customized log-log plot of $I(Q)$:

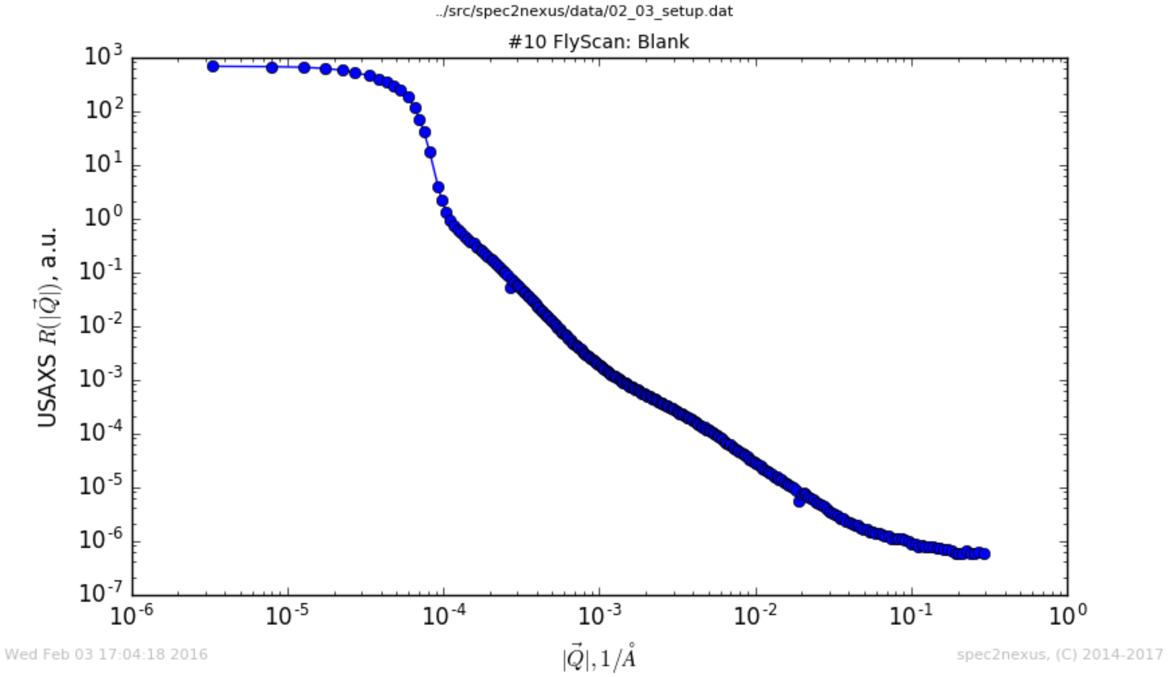


Fig. 9: USAXS *FlyScan*, handled by `USAXS_FlyScan_Plotter`

Usage

When a custom scan macro handler is written and installed using code similar to the *custom ascan* handling above:

```
def main():
    selector = spec2nexus.specplot.Selector()
    selector.add('ascan', Custom_Ascan)
    spec2nexus.specplot_gallery.main()


if __name__ == '__main__':
    main()
```

then the command line arugment handling from `spec2nexus.specplot_gallery.main()` can be accessed from the command line for help and usage information.

Usage:

```
user@localhost ~/.../spec2nexus/demo $ ./ascan.py
usage: ascan.py [-h] [-r] [-d DIR] paths [paths ...]
ascan.py: error: too few arguments
```

Help:

```
user@localhost ~/.../spec2nexus/demo $ ./ascan.py -h
usage: ascan.py [-h] [-r] [-d DIR] paths [paths ...]

read a list of SPEC data files (or directories) and plot images of all scans

positional arguments:
  paths                 SPEC data file name(s) or directory(s) with SPEC data
                        files

optional arguments:
  -h, --help            show this help message and exit
  -r                    sort images from each data file in reverse chronolgical
                        order
  -d DIR, --dir DIR     base directory for output (default:/home/prjemian/Documen
                        ts/eclipse/spec2nexus/demo)
```

2.1.8 spec2nexus.eznx

(Easy NeXus) support library for reading & writing NeXus HDF5 files using h5py

How to use spec2nexus.eznx

Here is a simple example to write a NeXus data file using eznx:

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

"""
Writes a simple NeXus HDF5 file using h5py with links.

This example is based on ``writer_2_1`` of the NeXus Manual:
http://download.nexusformat.org/doc/html/examples/h5py/index.html
"""

from spec2nexus import eznx


HDF5_FILE = "eznx_example.hdf5"

I_v_TTH_DATA = """
17.92608    1037
17.92558    2857
17.92508    23819
17.92458    49087
17.92408    66802
17.92358    66206
17.92308    64129
17.92258    56795
17.92208    29315
17.92158    6622
17.92108    1321
"""
# ---------------------------

tthData, countsData = zip(
```

(continues on next page)

(continued from previous page)

```
    *[map(float, _.split()) for _ in I_v_TTH_DATA.strip().splitlines()]
)

f = eznx.makeFile(HDF5_FILE)  # create the HDF5 NeXus file
f.attrs["default"] = "entry"

nxentry = eznx.makeGroup(f, "entry", "NXentry", default="data")
nxinstrument = eznx.makeGroup(nxentry, "instrument", "NXinstrument")
nxdetector = eznx.makeGroup(nxinstrument, "detector", "NXdetector")

tth = eznx.makeDataset(nxdetector, "two_theta", tthData, units="degrees")
counts = eznx.makeDataset(nxdetector, "counts", countsData, units="counts")

nxdata = eznx.makeGroup(
    nxentry,
    "data",
    "NXdata",
    signal=1,
    axes="two_theta",
    two_theta_indices=0,
)
eznx.makeLink(nxdetector, tth, nxdata.name + "/two_theta")
eznx.makeLink(nxdetector, counts, nxdata.name + "/counts")

f.close()  # be CERTAIN to close the file

# -----------------------------------------------------------------------------
# :author:    Pete R. Jemian
# :email:     prjemian@gmail.com
# :copyright: (c) 2014-2022, Pete R. Jemian
#
# Distributed under the terms of the Creative Commons Attribution 4.0 International Public License.
#
# The full license is in the file LICENSE.txt, distributed with this software.
# -----------------------------------------------------------------------------
```

The output of this code is an HDF5 file (binary). It has this structure:

```
    eznx_example.hdf5:NeXus data file
      @default = entry
      entry:NXentry
        @NX_class = NXentry
        @default = data
        data:NXdata
          @NX_class = NXdata
          @signal = counts
          @axes = two_theta
          @two_theta_indices = 0
          counts --> /entry/instrument/detector/counts
          two_theta --> /entry/instrument/detector/two_theta
        instrument:NXinstrument
          @NX_class = NXinstrument
          detector:NXdetector
            @NX_class = NXdetector
            counts:NX_FLOAT64[11] = __array
              @units = counts
```

(continues on next page)

(continued from previous page)

```
            @target = /entry/instrument/detector/counts
            __array = [1037.0, 2857.0, 23819.0, '...', 1321.0]
          two_theta:NX_FLOAT64[11] = __array
            @units = degrees
            @target = /entry/instrument/detector/two_theta
            __array = [17.926079999999999, 17.92558, 17.925080000000001, '...', 17.
↪92108]
```

NeXus HDF5 File Structure

The output of this code is an HDF5 file (binary). It has this general structure (indentation shows HDF5 groups, @ signs describe attributes of the preceding item):

```
hdf5_file:NeXus data file
   @default = S1
   S1:NXentry     (one NXentry for each scan)
      @default = data
      title = #S
      T or M: #T or #M
      comments: #C for entire scan
      date: #D
      scan_number: #S
      G:NXcollection
         @description = SPEC geometry arrays, meanings defined by SPEC␣
↪diffractometer support
         G0:NX_FLOAT64[] #G0
         G1:NX_FLOAT64[] #G1
         ...
      data:NXdata
         @description = SPEC scan data (content from #L and data lines)
         @signal = I0
         @axes = mr
         @mr_indices = 0
         Epoch:NX_FLOAT64[]
         I0:NX_FLOAT64[]         (last data column)
           @spec_name = I0
         mr:NX_FLOAT64[]         (first data column)
         ...
      metadata:NXcollection
         @description = SPEC metadata (UNICAT-style #H & #V lines)
         ARenc_0:NX_FLOAT64 = 0.0
         ...
      positioners:NXcollection
         @description = SPEC positioners (#P & #O lines)
         mr:NX_FLOAT64
         ...
```

APIs provided:

`spec2nexus.writer`

This is an internal library of the **spec2nexus** software. It is not expected that users of this package will need to call the writer module directly.

source code documentation

source code methods

addAttributes	add attributes to an h5py data item
makeFile	create and open an empty NeXus HDF5 file using h5py
makeDataset	create and write data to a dataset in the HDF5 file hierarchy
makeExternalLink	create an external link from sourceFile, sourcePath to targetPath in hdf5FileObject
makeGroup	create a NeXus group
openGroup	open or create the NeXus/HDF5 group, return the object
makeLink	create an internal NeXus (hard) link in an HDF5 file
read_nexus_field	get a dataset from the HDF5 parent group
read_nexus_group_fields	return the fields in the NeXus group as a dict(name=dataset)
write_dataset	write to the NeXus/HDF5 dataset, create it if necessary, return the object

source code documentation

(Easy NeXus) support reading & writing NeXus HDF5 files using h5py

predecessor NeXus h5py example code: `my_lib.py`[1]

Dependencies

- h5py: interface to HDF5 file format

Exceptions raised

- None

Example

```
root = eznx.makeFile('test.h5', creator='eznx', default='entry')
nxentry = eznx.makeGroup(root, 'entry', 'NXentry', default='data')
ds = eznx.write_dataset(nxentry, 'title', 'simple test data')
nxdata = eznx.makeGroup(nxentry, 'data', 'NXdata', signal='counts', axes='tth', tth_
↪indices=0)
ds = eznx.write_dataset(nxdata, 'tth', [10.0, 10.1, 10.2, 10.3], units='degrees')
ds = eznx.write_dataset(nxdata, 'counts', [1, 50, 1000, 5], units='counts', axes="tth
↪")
root.close()
```

The resulting (binary) data file has this structure:

[1] http://download.nexusformat.org/doc/html/examples/h5py/index.html#mylib-support-module

```
test.h5:NeXus data file
  @creator = eznx
  @default = 'entry'
  entry:NXentry
    @NX_class = NXentry
    @default = 'data'
    title:NX_CHAR = simple test data
    data:NXdata
      @NX_class = NXdata
      @signal = 'counts'
      @axes = 'tth'
      @tth_indices = 0
      counts:NX_INT64[4] = [1, 50, 1000, 5]
        @units = counts
        @axes = tth
      tth:NX_FLOAT64[4] = [10.0, 10.1, 10.199999999999999, 10.300000000000001]
        @units = degrees
```

Classes and Methods

`spec2nexus.eznx.`**`addAttributes`**(*parent*, ***attr*)

add attributes to an h5py data item

Parameters

- **parent** (*obj*) – h5py parent object
- **attr** (*dict*) – optional dictionary of attributes

`spec2nexus.eznx.`**`makeDataset`**(*parent*, *name*, *data=None*, ***attr*)

create and write data to a dataset in the HDF5 file hierarchy

Any named parameters in the call to this method will be saved as attributes of the dataset.

Parameters

- **parent** (*obj*) – parent group
- **name** (*str*) – valid NeXus dataset name
- **data** (*obj*) – the information to be written
- **attr** (*dict*) – optional dictionary of attributes

Returns h5py dataset object

`spec2nexus.eznx.`**`makeExternalLink`**(*hdf5FileObject*, *sourceFile*, *sourcePath*, *targetPath*)

create an external link from sourceFile, sourcePath to targetPath in hdf5FileObject

Parameters

- **hdf5FileObject** (*obj*) – open HDF5 file object
- **sourceFile** (*str*) – file containing existing HDF5 object at sourcePath
- **sourcePath** (*str*) – path to existing HDF5 object in sourceFile
- **targetPath** (*str*) – full node path to be created in current open HDF5 file, such as `/entry/data/data`

Note: Since the object retrieved is in a different file, its ".file" and ".parent" properties will refer to objects in that file, not the file in which the link resides.

See http://www.h5py.org/docs-1.3/guide/group.html#external-links

This routine is provided as a reminder how to do this simple operation.

`spec2nexus.eznx.`**`makeFile`**(*filename*, ***attr*)

create and open an empty NeXus HDF5 file using h5py

Any named parameters in the call to this method will be saved as attributes of the root of the file. Note that `**attr` is a dictionary of named parameters.

Parameters

- **`filename`** (*`str`*) – valid file name
- **`attr`** (*`dict`*) – optional dictionary of attributes

Returns h5py file object

`spec2nexus.eznx.`**`makeGroup`**(*parent*, *name*, *nxclass*, ***attr*)

create a NeXus group

Any named parameters in the call to this method will be saved as attributes of the group. Note that `**attr` is a dictionary of named parameters.

Parameters

- **`parent`** (*`obj`*) – parent group
- **`name`** (*`str`*) – valid NeXus group name
- **`nxclass`** (*`str`*) – valid NeXus class name
- **`attr`** (*`dict`*) – optional dictionary of attributes

Returns h5py group object

`spec2nexus.eznx.`**`makeLink`**(*parent*, *sourceObject*, *targetName*)

create an internal NeXus (hard) link in an HDF5 file

Parameters

- **`parent`** (*`obj`*) – parent group of source
- **`sourceObject`** (*`obj`*) – existing HDF5 object
- **`targetName`** (*`str`*) – HDF5 node path to be created, such as `/entry/data/data`

`spec2nexus.eznx.`**`openGroup`**(*parent*, *name*, *nx_class*, ***attr*)

open or create the NeXus/HDF5 group, return the object

Parameters

- **`parent`** (*`obj`*) – h5py parent object
- **`name`** (*`str`*) – valid NeXus group name to open or create
- **`nxclass`** (*`str`*) – valid NeXus class name (base class or application definition)
- **`attr`** (*`dict`*) – optional dictionary of attributes

`spec2nexus.eznx.`**`read_nexus_field`**(*parent*, *dataset_name*, *astype=None*)

get a dataset from the HDF5 parent group

Parameters

- **`parent`** (*`obj`*) – h5py parent object
- **`dataset_name`** (*`str`*) – name of the dataset (NeXus field) to be read
- **`astype`** (*`obj`*) – option to return as different data type

`spec2nexus.eznx.`**`read_nexus_group_fields`**(*parent*, *name*, *fields*)

return the fields in the NeXus group as a dict(name=dataset)

This routine provides a mass way to read a directed list of datasets (NeXus fields) in an HDF5 group.

Parameters

- **`parent`** (*`obj`*) – h5py parent object
- **`name`** (*`str`*) – name of the group containing the fields
- **`fields`** (*`[name]`*) – list of field names to be read

Returns dictionary of {name:dataset}

Raises **`KeyError`** – if a field is not found

`spec2nexus.eznx.`**`write_dataset`**(*parent*, *name*, *data*, ***attr*)

write to the NeXus/HDF5 dataset, create it if necessary, return the object

Parameters

- **`parent`** (*`obj`*) – h5py parent object
- **`name`** (*`str`*) – valid NeXus dataset name to write
- **`data`** (*`obj`*) – the information to be written
- **`attr`** (*`dict`*) – optional dictionary of attributes

2.1.9 `spec2nexus.plugin`

An extensible plug-in architecture is used to handle the different possible control line control lines (such as **#F**, **#E**, **#S**, . . .) in a SPEC data file.

A SPEC *control line* provides metadata about the SPEC scan or SPEC data file.

Plugins can be used to parse or ignore certain control lines in SPEC data files. Through this architecture, it is possible to support custom control lines, such as **#U** (SPEC standard control line for any user data). One example is support for the *UNICAT-style* of metadata provided in the scan header.

Plugins are now used to handle all control lines in `spec2nexus.spec`. Any control line encountered but not recognized will be placed as text in a NeXus **NXnote** group named `unrecognized_NNN` (where `NNN` is from 1 to the maximum number of unrecognized control lines).

Supplied spec plugin modules

These plugin modules are supplied:

`spec2nexus.plugins.spec_common`
`spec2nexus.plugins.fallback`
`spec2nexus.plugins.apstools_specwriter`

Continued on next page

Table 7 – continued from previous page

spec2nexus.plugins.unicat
spec2nexus.plugins.uim
spec2nexus.plugins.uxml
spec2nexus.plugins.XPCS

XPCS plugin

apstools SpecWriterCallback metadata plugin

Looks for #MD control line control lines. These lines contain metadata supplied to the bluesky RunEngine and recorded during the execution of a scan. The data are stored in a dictionary of each scan: scan.MD. If there are no #MD control lines, then scan.MD does not exist.

see https://prjemian.github.io/spec2nexus/source/_filewriters.html#apstools.filewriters.SpecWriterCallback

---

Fallback plugin

SPEC standard plugin

UIM plugin

unicat plugin

#UXML: UXML metadata plugin

Looks for #UXML control line control lines. These lines contain metadata written as XML structures and formatted according to the supplied XML Schema uxml.xsd in the same directory as the uxml.py plugin. The lines which comprise the XML are written as a list in each scan: scan.UXML. If there are no #UXML control lines, then scan.UXML does not exist.

Once the scan has been fully read scan.UXML is converted into an XML document structure (using the *lxml.etree* package) which is stored in scan.UXML_root. The structure is validated against the XML Schema uxml.xsd. If invalid, the error message is reported by raising a UXML_Error python exception.

A fully-validated structure can be written using the Writer class. The UXML metadata is written to the scan's NXentry group as subgroup named UXML with NeXus base class NXnote. The hierarchy within this UXML is defined from the content provided in the SPEC scan.

Please consult the XML Schema file for the rules governing the use of #UXML in a SPEC data file: * uxml.xsd

---

Writing a custom plugin

While **spec2nexus** provides a comprehensive set of plugins to handle the common SPEC control line control lines, custom control lines are used at many facilities to write additional scan data and scan metadata into the SPEC data file. Custom plugins are written to process these additions.

How to write a custom plugin module

The code to write plugins has changed with release 2021.0.0.

The changes are summarized in the section below titled *Changes in plugin format with release 2021.0.0*.

Sections

A custom plugin module for `spec2nexus.spec` is provided in a python module (Python source code file). In this custom plugin module are subclasses for each *new control line* to be supported. An exception will be raised if a custom plugin module tries to provide support for an existing control line.

Load a plugin module

Control line handling plugins for *spec2nexus* will automatically register themselves when their module is imported. Be sure that you call *get_plugin_manager()* **before** you `import` your plugin code. This step sets up the plugin manager to automatically register your new plugin.

```
import spec2nexus.plugin
import spec2nexus.spec

# get the plugin manager BEFORE you import any custom plugins
manager = plugin.get_plugin_manager()

import MY_PLUGIN_MODULE
# ... more if needed ...

# read a SPEC data file, scan 5
spec_data_file = spec2nexus.spec.SpecDataFile("path/to/spec/datafile")
scan5 = spec_data_file.getScan(5)
```

Write a plugin module

Give the custom plugin module a name ending with `.py`. As with any Python module, the name must be unique within a directory. If the plugin is not in your working directory, there must be a `__init__.py` file in the same directory (even if that file is empty) so that your plugin module can be loaded with `import <MODULE>`.

Plugin module setup

The `six` package

The `six` package is used to make our plugins run with either Python 2.7 or Python 3.5+.

Please view the existing plugins in `spec_common` for examples. The custom plugin module should contain, at minimum one subclass of *`spec2nexus.plugin.ControlLineHandler`* which is decorated with `@six.add_metaclass(spec2nexus.plugin.AutoRegister)`. The `add_metaclass` decorator allows our custom ControlLineHandlers to register themselves when their module is imported. A custom plugin module can contain many such handlers, as needs dictate.

Useful `import`

It is also useful to import the *`strip_first_word()`* utility method.

These imports are necessary to to write plugins for *spec2nexus*:

```
import six
from spec2nexus.plugin import AutoRegister
from spec2nexus.plugin import ControlLineHandler
from spec2nexus.utils import strip_first_word
```

regular expressions

There are several regular expression testers available on the web. Try this one, for example: http://regexpal.com/

Attribute: ‘‘key‘‘ (required)

Each subclass must define key `key` as a regular expression match for the control line key. It is possible to override any of the supplied plugins for scan control line control lines. Caution is advised to avoid introducing instability.

Attribute: ‘‘scan_attributes_defined‘‘ (optional)

If your plugin creates any attributes to the `spec2nexus.spec.SpecDataScan` object (such as the hypotetical `scan.hdf5_path` and `scan.hdf5_file`), you declare the new attributes in the `scan_attributes_defined` list. Such as this:

```
scan_attributes_defined = ['hdf5_path', 'hdf5_file']
```

Method: ‘‘process()‘‘ (required)

Each subclass must also define a `process()` method to process the control line. A `NotImplementedError` exception is raised if `key` is not defined.

Method: ‘‘match_key()‘‘ (optional)

For difficult regular expressions (or other situations), it is possible to replace the function that matches for a particular control line key. Override the handler's `match_key()` method. For more details, see the section *Custom key match function*.

Method: ‘‘postprocess()‘‘ (optional)

For some types of control lines, processing can only be completed *after* all lines of the scan have been read. In such cases, add a line such as this to the `process()` method:

```
scan.addPostProcessor(self.key, self.postprocess)
```

(You *could* replace `self.key` here with some other text. If you do, make sure that text will be unique as it is used internally as a python dictionary key.) Then, define a `postprocess()` method in your handler:

```
def postprocess(self, scan, *args, **kws):
    # handle your custom info here
```

See section *Postprocessing* below for more details. See `spec2nexus.plugins.spec_common` for many examples.

Method: ‘‘writer()‘‘ (optional)

Writing a NeXus HDF5 data file is one of the main goals of the *spec2nexus* package. If you intend data from your custom control line handler to end up in the HDF5 data file, add a line such as this to either the `process()` or `postprocess()` method:

```
scan.addH5writer(self.key, self.writer)
```

Then, define a `writer()` method in your handler. Here's an example:

```
def writer(self, h5parent, writer, scan, nxclass=None, *args, **kws):
    """Describe how to store this data in an HDF5 NeXus file"""
    desc='SPEC positioners (#P & #O lines)'
    group = makeGroup(h5parent, 'positioners', nxclass, description=desc)
    writer.save_dict(group, scan.positioner)
```

See section *Custom HDF5 writer* below for more details.

Full Example: #PV control line

Consider a SPEC data file (named `pv_data.txt`) with the contrived example of a **#PV** control line that associates a mnemonic with an EPICS process variable (PV). Suppose we take this control line content to be two words (text with no whitespace):

```
#F pv_data.txt
#E 1454539891
#D Wed Feb 03 16:51:31 2016
#C pv_data.txt  User = spec2nexus
#O0 USAXS.a2rp  USAXS.m2rp  USAXS.asrp  USAXS.msrp  mr  unused37  mst  ast
#O1 msr  asr  unused42  unused43  ar  ay  dy  un47

#S 1  ascan  mr 10.3467 10.3426  30 0.1
#D Wed Feb 03 16:52:03 2016
#T 0.1  (seconds)
#P0 3.5425 6.795 7.7025 5.005 10.34465 0 0 0
#P1 7.6 17.17188 -8.67896 -0.351 10.318091 0 18.475664 0
#C tuning USAXS motor mr
```

(continues on next page)

(continued from previous page)

```
#PV mr ioc:m1
#PV ay ioc:m2
#PV dy ioc:m3
#N 18
#L mr    ay  dy  ar_enc  pd_range  pd_counts  pd_rate  pd_curent  I0_gain  I00_gain ␣
↪Und_E  Epoch  seconds  I00  USAXS_PD  TR_diode  I0  I0
10.34665  0.000 18.476 10.318091 1 5 481662 0.000481658 1e+07 1e+09 18.172565 33.037␣
↪0.1 199 2 1 114 114
10.34652  0.000 18.476 10.318091 1 5 481662 0.000481658 1e+07 1e+09 18.172565 33.294␣
↪0.1 198 2 1 139 139
10.34638  0.000 18.476 10.318091 1 5 481662 0.000481658 1e+07 1e+09 18.172565 33.553␣
↪0.1 198 2 1 181 181
10.34625  0.000 18.476 10.318091 1 5 481662 0.000481658 1e+07 1e+09 18.172565 33.952␣
↪0.1 198 2 1 274 274
10.34278  0.000 18.476 10.318091 1 5 481662 0.000481658 1e+07 1e+09 18.172309 41.621␣
↪0.1 198 2 1 232 232
10.34265  0.000 18.476 10.318091 1 5 481662 0.000481658 1e+07 1e+09 18.172565 41.867␣
↪0.1 199 2 1 159 159
#C Wed Feb 03 16:52:14 2016.  removed many data rows for this example.
```

A plugin (named `pv_plugin.py`) to handle the `#PV` control lines could be written as:

```python
from collections import OrderedDict
import six
from spec2nexus.plugin import AutoRegister
from spec2nexus.plugin import ControlLineHandler
from spec2nexus.utils import strip_first_word

@six.add_metaclass(AutoRegister)
class PV_ControlLine(ControlLineHandler):
    '''**#PV** -- EPICS PV associates mnemonic with PV'''

    key = '#PV'
    scan_attributes_defined = ['EPICS_PV']

    def process(self, text, spec_obj, *args, **kws):
        args = strip_first_word(text).split()
        mne = args[0]
        pv = args[1]
        if not hasattr(spec_obj, "EPICS_PV"):
            # use OrderedDict since it remembers the order we found these
            spec_obj.EPICS_PV = OrderedDict()
        spec_obj.EPICS_PV[mne] = pv
```

When the scan parser encounters the **#PV** lines in our SPEC data file, it will call this `process()` code with the full text of the line and the spec scan object where this data should be stored. We will choose to store this (following the pattern of other data names in `SpecDataFileScan`) as `scan_obj.EPICS_PV` using a dictionary.

It is up to the user what to do with the `scan_obj.EPICS_PV` data. We will not consider the `write()` method in this example. (We will not write this infromation to a NeXus HDF5 file.)

We can then write a python program (named `pv_example.py`) that will load the data file and interpret it using our custom plugin:

```python
import spec2nexus.plugin
import spec2nexus.spec

```

(continues on next page)

(continued from previous page)

```
# call get_plugin_manager() BEFORE you import any custom plugins
manager = spec2nexus.plugin.get_plugin_manager()

# show our plugin is not loaded
print("known: ", "#PV" in manager.registry) # expect False

import pv_plugin
# show that our plugin is registered
print("known: ", "#PV" in manager.registry) # expect True

# read a SPEC data file, scan 1
spec_data_file = spec2nexus.spec.SpecDataFile("pv_data.txt")
scan = spec_data_file.getScan(1)

# Do we have our PV data?
print(hasattr(scan, "EPICS_PV"))     # expect True
print(scan.EPICS_PV)
```

The output of our program:

```
known:  False
known:  True
False
True
OrderedDict([('mr', 'ioc:m1'), ('ay', 'ioc:m2'), ('dy', 'ioc:m3')])
```

Example to ignore a #Y control line

Suppose a control line in a SPEC data file must be ignored. For example, suppose a SPEC file contains this control line: `#Y 1 2 3 4 5`. Since there is no standard handler for this control line, we create one that ignores processing by doing nothing:

```
import six
from spec2nexus.plugin import AutoRegister
from spec2nexus.plugin import ControlLineHandler

@six.add_metaclass(AutoRegister)
class Ignore_Y_ControlLine(ControlLineHandler):
    '''
    **#Y** -- as in ``#Y 1 2 3 4 5``

    example: ignore any and all #Y control lines
    '''

    key = '#Y'

    def process(self, text, spec_obj, *args, **kws):
        pass # do nothing
```

Postprocessing

Sometimes, it is necessary to defer a step of processing until after the complete scan data has been read. One example is for 2-D or 3-D data that has been acquired as a vector rather than matrix. The matrix must be constructed only after all the scan data has been read. Such postprocessing is handled in a method in a plugin file. The postprocessing method is registered from the control line handler by calling the `addPostProcessor()` method of the `spec_obj` argument received by the handler's `process()` method. A key name[1] is supplied when registering to avoid registering this same code more than once. The postprocessing function will be called with the instance of `SpecDataFileScan` as its only argument.

An important role of the postprocessing is to store the result in the scan object. It is important not to modify other data in the scan object. Pick an attribute named similarly to the plugin (e.g., MCA configuration uses the **MCA** attribute, UNICAT metadata uses the **metadata** attribute, . . .) This attribute will define where and how the data from the plugin is available. The `writer()` method (see *below*) is one example of a user of this attribute.

Example postprocessing

Consider the **#U** control line example above. For some contrived reason, we wish to store the sum of the numbers as a separate number, but only after all the scan data has been read. This can be done with the simple expression:

```
spec_obj.U_sum = sum(spec_obj.U)
```

To build a postprocessing method, we write:

```
def contrived_summation(scan):
    '''
    add up all the numbers in the #U line

    :param SpecDataFileScan scan: data from a single SPEC scan
    '''
    scan.U_sum = sum(scan.U)
```

To register this postprocessing method, place this line in the `process()` of the handler:

```
spec_obj.addPostProcessor('contrived_summation', contrived_summation)
```

Summary Example Custom Plugin with postprocessing

Gathering all parts of the examples above, the custom plugin module is:

```
import six
from spec2nexus.plugin import AutoRegister
from spec2nexus.plugin import ControlLineHandler
from spec2nexus.utils import strip_first_word

@six.add_metaclass(AutoRegister)
class User_ControlLine(ControlLineHandler):
    '''**#U** -- User data (#U user1 user2 user3)'''

    key = '#U'

    def process(self, text, spec_obj, *args, **kws):
        args = strip_first_word(text).split()
        user1 = float(args[0])
        user2 = float(args[1])
        user3 = float(args[2])
        spec_obj.U = [user1, user2, user3]
```

(continues on next page)

[1] The key name must be unique amongst all postprocessing functions. A good choice is the name of the postprocessing function itself.

(continued from previous page)

```
        spec_obj.addPostProcessor('contrived_summation', contrived_summation)


def contrived_summation(scan):
    '''
    add up all the numbers in the #U line

    :param SpecDataFileScan scan: data from a single SPEC scan
    '''
    scan.U_sum = sum(scan.U)


@six.add_metaclass(AutoRegister)
class Ignore_Y_ControlLine(ControlLineHandler):
    '''**#Y** -- as in ``#Y 1 2 3 4 5``'''

    key = '#Y'

    def process(self, text, spec_obj, *args, **kws):
        pass
```

Custom HDF5 writer

A custom HDF5 writer method defines how the data from the *plugin* will be written to the HDF5+NeXus data file. The writer will be called with several arguments:

h5parent: *obj* : the HDF5 group that will hold this plugin's data

writer: *obj* : instance of `spec2nexus.writer.Writer` that manages the content of the HDF5 file

scan: *obj* : instance of `spec2nexus.spec.SpecDataFileScan` containing this scan's data

nxclass: *str* : (optional) name of NeXus base class to be created

Since the file is being written according to the NeXus data standard[2], use the NeXus base classes[3] as references for how to structure the data written by the custom HDF5 writer.

One responsibility of a custom HDF5 writer method is to create *unique* names for every object written in the *h5parent* group. Usually, this will be a *NXentry*[4] group. You can determine the NeXus base class of this group using code such as this:

```
>>> print h5parent.attrs['NX_class']
<<< NXentry
```

If your custom HDF5 writer must create group and you are uncertain which base class to select, it is recommended to use a **NXcollection**[5] (an unvalidated catch-all base class) which can store any content. But, you are encouraged to find one of the other NeXus base classes that best fits your data. Look at the source code of the supplied plugins for examples.

The writer uses the *`spec2nexus.eznx`* module to create and write the various parts of the HDF5 file.

Here is an example `writer()` method from the `spec2nexus.plugins.unicat` module:

[2] http://nexusformat.org
[3] http://download.nexusformat.org/doc/html/classes/base_classes/
[4] http://download.nexusformat.org/doc/html/classes/base_classes/NXentry.html
[5] http://download.nexusformat.org/doc/html/classes/base_classes/NXcollection.html

```
def writer(self, h5parent, writer, scan, nxclass=None, *args, **kws):
    '''Describe how to store this data in an HDF5 NeXus file'''
    if hasattr(scan, 'metadata') and len(scan.metadata) > 0:
        desc='SPEC metadata (UNICAT-style #H & #V lines)'
        group = eznx.makeGroup(h5parent, 'metadata', nxclass, description=desc)
        writer.save_dict(group, scan.metadata)
```

Custom key match function

The default test that a given line matches a specific *`spec2nexus.plugin.ControlLineHandler`* subclass is to use a regular expression match.

```
def match_key(self, text):
    '''default regular expression match, based on self.key'''
    t = re.match(self.key, text)
    if t is not None:
        if t.regs[0][1] != 0:
            return True
    return False
```

In some cases, that may prove tedious or difficult, such as when testing for a floating point number with optional preceding white space at the start of a line. This is typical for data lines in a scan or continued lines from an MCA spectrum. in such cases, the handler can override the `match_key()` method. Here is an example from `SPEC_DataLine`:

```
def match_key(self, text):
    '''
    Easier to try conversion to number than construct complicated regexp
    '''
    try:
        float( text.strip().split()[0] )
        return True
    except ValueError:
        return False
```

Summary Requirements for custom plugin

- file can go in your working directory or any directory that has `__init__.py` file
- multiple control line handlers can go in a single file
- for each control line:
  - subclass *`spec2nexus.plugin.ControlLineHandler`*
  - add `@six.add_metaclass(AutoRegister)` decorator to auto-register the plugin
  - import the module you defined (FIXME: check this and revise)
  - identify the control line pattern
  - define `key` with a regular expression to match[6]
    - `key` is used to identify control line handlers

[6] It is possible to override the default regular expression match in the subclass with a custom match function. See the `match_key()` method for an example.

        - redefine existing supported control line control lines to replace supplied behavior (use caution!)
        - Note: `key="scan data"` is used to process the scan data: `spec2nexus.plugins.spec_common.SPEC_DataLine()`
    - define `process()` to handle the supplied text
    - define `writer()` to write the in-memory data structure from this plugin to HDF5+NeXus data file
    - (optional) define `match_key()` to override the default regular expression to match the key
- for each postprocessing function:
    - write the function
    - register the function with spec_obj.addPostProcessor(key_name, the_function) in the handler's `process()`

Changes in plugin format with release 2021.0.0

With release *2021.0.0*, the code to setup plugins has changed. The new code allows all plugins in a module to auto-register themselves *as long as the module is imported*. **All** custom plugins must be modified and import code revised to work with new system. See the `spec2nexus.plugins.spec_common` source code for many examples.

- SAME: The basics of writing the plugins remains the same.
- CHANGED: The method of registering the plugins has changed.
- CHANGED: The declaration of each plugin has changed.
- CHANGED: The name of each plugin file has been relaxed.
- CHANGED: Plugin files do not have to be in their own directory.
- REMOVED: The `SPEC2NEXUS_PLUGIN_PATH` environment variable has been eliminated.

Footnotes

Overview of the supplied spec plugins

Plugins for these control lines[1] are provided in **spec2nexus**:

`spec2nexus.plugins.spec_common.SPEC_File`
`spec2nexus.plugins.spec_common.SPEC_Epoch`
`spec2nexus.plugins.spec_common.SPEC_Date`
`spec2nexus.plugins.spec_common.SPEC_Comment`
`spec2nexus.plugins.spec_common.SPEC_Geometry`

Continued on next page

[1] Compare this list with *Control lines (keys) defined by SPEC*

Table 8 – continued from previous page

spec2nexus.plugins.spec_common.SPEC_NormalizingFactor
spec2nexus.plugins.spec_common.SPEC_CounterNames
spec2nexus.plugins.spec_common.SPEC_CounterMnemonics
spec2nexus.plugins.spec_common.SPEC_Labels
spec2nexus.plugins.spec_common.SPEC_Monitor
spec2nexus.plugins.spec_common.SPEC_NumColumns
spec2nexus.plugins.spec_common.SPEC_PositionerNames
spec2nexus.plugins.spec_common.SPEC_PositionerMnemonics
spec2nexus.plugins.spec_common.SPEC_Positioners
spec2nexus.plugins.spec_common.SPEC_HKL
spec2nexus.plugins.spec_common.SPEC_Scan
spec2nexus.plugins.spec_common.SPEC_CountTime
spec2nexus.plugins.spec_common.SPEC_UserReserved
spec2nexus.plugins.spec_common.SPEC_TemperatureSetPoint
spec2nexus.plugins.spec_common.SPEC_DataLine
spec2nexus.plugins.spec_common.SPEC_MCA
spec2nexus.plugins.spec_common.SPEC_MCA_Array
spec2nexus.plugins.spec_common.SPEC_MCA_Calibration
spec2nexus.plugins.spec_common.SPEC_MCA_ChannelInformation
spec2nexus.plugins.spec_common.SPEC_MCA_CountTime
spec2nexus.plugins.spec_common.SPEC_MCA_RegionOfInterest
spec2nexus.plugins.fallback.UnrecognizedControlLine
spec2nexus.plugins.unicat.UNICAT_MetadataMnemonics
spec2nexus.plugins.unicat.UNICAT_MetadataValues
spec2nexus.plugins.uim.UIM_generic
spec2nexus.plugins.XPCS.XPCS_VA
spec2nexus.plugins.XPCS.XPCS_VD

Continued on next page

Table 8 – continued from previous page

spec2nexus.plugins.XPCS.XPCS_VE

source code documentation

define the plug-in architecture

Use *spec2nexus.plugin.ControlLineHandler* as a metaclass to create a plugin handler class for each SPEC control line. In each such class, it is necessary to:

- define a string value for the `key` (class attribute)
- override the definition of `process()`

It is optional to:

- define `postprocess()`
- define `writer()`
- define `match_key()`

Classes

ControlLineHandler	base class for SPEC data file control line handler plugins
PluginManager()	Manage the set of SPEC data file control line plugins

Exceptions

DuplicateControlLineKey	This control line key regular expression has been used more than once.
DuplicateControlLinePlugin	This control line handler has been used more than once.
DuplicatePlugin	This plugin file name has been used more than once.
PluginBadKeyError	The plugin 'key' value is not acceptable.
PluginDuplicateKeyError	This plugin key has been used before.
PluginKeyNotDefined	Must define 'key' in class declaration.
PluginProcessMethodNotDefined	Must define 'process()' method in class declaration.

class spec2nexus.plugin.**AutoRegister**(**args*)

plugin to handle a single control line in a SPEC data file

This class is a metaclass to auto-register plugins to handle various parts of a SPEC data file. See `spec_common` for many examples.

Parameters **key** (*str*) – regular expression to match a control line key, up to the first space

Returns None

class spec2nexus.plugin.**ControlLineHandler**

base class for SPEC data file control line handler plugins

define one ControlLineHandler class for each different type of control line

Parameters

- **key** (*str*) – regular expression to match a control line key, up to the first space

- **scan_attributes_defined** (*[str]*) – list of scan attributes defined in this class

Returns None

EXAMPLE of `match_key` method:

Declaration of the `match_key` method is optional in a subclass. This is used to test a given line from a SPEC data file against the `key` of each `ControlLineHandler`.

If this method is defined in the subclass, it will be called instead of *`match_key()`*. This is the example used by `SPEC_DataLine`:

```
def match_key(self, text):
    try:
        float( text.strip().split()[0] )
        return True
    except ValueError:
        return False
```

postprocess (*header*, **args*, ***kws*)
optional: additional processing deferred until *after* data file has been read

process (*text*, *spec_file_obj*, **args*, ***kws*)
required: handle this line from a SPEC data file

writer (*h5parent*, *writer*, *scan*, *nxclass=None*, **args*, ***kws*)
optional: Describe how to store this data in an HDF5 NeXus file

exception spec2nexus.plugin.**DuplicateControlLineKey**
This control line key regular expression has been used more than once.

exception spec2nexus.plugin.**DuplicateControlLinePlugin**
This control line handler has been used more than once.

exception spec2nexus.plugin.**DuplicatePlugin**
This plugin file name has been used more than once.

exception spec2nexus.plugin.**PluginBadKeyError**
The plugin ‘key’ value is not acceptable.

exception spec2nexus.plugin.**PluginDuplicateKeyError**
This plugin key has been used before.

exception spec2nexus.plugin.**PluginException**
parent exception for this module

exception spec2nexus.plugin.**PluginKeyNotDefined**
Must define ‘key’ in class declaration.

class spec2nexus.plugin.**PluginManager**
Manage the set of SPEC data file control line plugins

Class Methods

get(key)	return the handler identified by key or None
getKey(spec_data_file_line)	Find the key that matches this line in a SPEC data file.
load_plugins()	load all spec2nexus plugin modules
match_key(text)	test if any handler’s key matches text

Continued on next page

Table 11 – continued from previous page

process(key, *args, **kw)	pick the control line handler by key and call its process() method
register_control_line_handler(handler)	auto-registry of all AutoRegister plugins

get (*key*)
return the handler identified by key or None

getKey (*spec_data_file_line*)
Find the key that matches this line in a SPEC data file. Return None if not found.

Parameters **spec_data_file_line** (`str`) – one line from a SPEC data file

load_plugins ()
load all spec2nexus plugin modules

called from `spec2nexus.plugin.get_plugin_manager()`

match_key (*text*)
test if any handler's key matches text

Parameters **text** (`str`) – first word on the line, up to but not including the first whitespace

Returns key or None

Applies a regular expression match using each handler's `key` as the regular expression to match with `text`.

process (*key*, **args*, ***kw*)
pick the control line handler by key and call its process() method

register_control_line_handler (*handler*)
auto-registry of all AutoRegister plugins

Called from AutoRegister.__init__

exception spec2nexus.plugin.**PluginProcessMethodNotDefined**
Must define 'process()' method in class declaration.

spec2nexus.plugin.**get_plugin_manager** ()
get the instance of the plugin_manager (a singleton)

Create instance of PluginManager() if necessary. Also,

2.1.10 Common Methods: `spec2nexus.utils`

source code documentation

(internal library) common methods used in **spec2nexus** modules

clean_name(key)	create a name that is allowed by both HDF5 and NeXus rules
iso8601(date)	convert SPEC time (example: Wed Nov 03 13:39:34 2010) into ISO8601 string
strip_first_word(line)	return everything after the first space on the line from the spec data file
sanitize_name(group, key)	make name that is allowed by HDF5 and NeXus rules
reshape_data(scan_data, scan_shape)	Shape scan data from raw to different dimensionality

spec2nexus.utils.**clean_name**(*key*)

create a name that is allowed by both HDF5 and NeXus rules

Parameters **key** (*str*) – identifying string from SPEC data file

See http://download.nexusformat.org/doc/html/datarules.html

The "sanitized" name fits this regexp:

```
[A-Za-z_][\w_]*
```

An easier expression might be: `[\w_]*` but this will not pass the rule that valid NeXus group or field names cannot start with a digit.

spec2nexus.utils.**iso8601**(*date*)

convert SPEC time (example: Wed Nov 03 13:39:34 2010) into ISO8601 string

Parameters **date** (*str*) – time string from SPEC data file

Example

SPEC Wed Nov 03 13:39:34 2010

ISO8601 2010-11-03T13:39:34

SPOCK 09/15/17 04:39:10

ISO8601 2017-09-15T04:39:10

spec2nexus.utils.**reshape_data**(*scan_data*, *scan_shape*)

Shape scan data from raw to different dimensionality

Some SPEC macros collect data in a mesh or grid yet report the data as a 1-D sequence of observations. For further processing (such as plotting), the scan data needs to be reshaped according to its intended dimensionality.

modified from nexpy.readers.readspec.reshape_data

spec2nexus.utils.**sanitize_name**(*group*, *key*)

make name that is allowed by HDF5 and NeXus rules

Note **deprecated** use *clean_name()* instead (`group` is never used)

Parameters

- **group** (*str*) – unused
- **key** (*str*) – identifying string from SPEC data file

See http://download.nexusformat.org/doc/html/datarules.html

sanitized name fits this regexp:

```
[A-Za-z_][\w_]*
```

An easier expression might be: `[\w_]*` but this will not pass the rule that valid names cannot start with a digit.

spec2nexus.utils.**split_column_labels**(*text*)

SPEC labels may contain one space

spec2nexus.utils.**strip_first_word**(*line*)

return everything after the first space on the line from the spec data file

2.1.11 `spec2nexus.scanf`

Simple scanf-implementation. This module provides an easy way to parse simple formatted strings. It works similar to the version C programmers are used to.

source code documentation

Small scanf-implementation.

- Created by Henning Schroeder on Mon, 12 Feb 2007
- PSF license

Python has powerful regular expressions but sometimes they are totally overkill when you just want to parse a simple-formatted string. C programmers use the scanf-function for these tasks (see link below).

This implementation of scanf translates the simple scanf-format into regular expressions. Unlike C you can be sure that there are no buffer overflows possible.

source: http://code.activestate.com/recipes/502213-simple-scanf-implementation/

For more information see:

- http://www.python.org/doc/current/lib/node49.html
- http://en.wikipedia.org/wiki/Scanf

`spec2nexus.scanf.`**`scanf`**(*fmt*, *s=None*)

scanf supports the following formats:

format	description
%c	One character
%5c	5 characters
%d	int value
%7d	int value with length 7
%f	float value
%o	octal value
%X, %x	hex value
%s	string terminated by whitespace

Examples: >>> scanf("%s - %d errors, %d warnings", "/usr/sbin/sendmail - 0 errors, 4 warnings") ('/usr/sbin/sendmail', 0, 4) >>> scanf("%o %x %d", "0123 0x123 123") (66, 291, 123)

If the parameter s is a file-like object, s.readline is called. If s is not specified, stdin is assumed.

The function returns a tuple of found values or None if the format does not match.

2.1.12 `spec2nexus.singletons`

This is an internal library of the **spec2nexus** software. It is not expected that users of this package will need to call the *singletons* module directly.

source code documentation

singletons: Python 2 and 3 Compatible Version

see http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python

USAGE:

```
class Logger(Singleton):
    pass
```

class spec2nexus.singletons.**Singleton**

Public interface

2.1.13 Installation

Released versions of spec2nexus are available on PyPI.

If you have `pip` installed, then you can install:

```
$ pip install spec2nexus
```

If you are using Anaconda Python and have `conda` installed, then you can install with either of these:

```
$ conda install -c aps-anl-tag spec2nexus
$ conda install -c aps-anl-dev spec2nexus
$ conda install -c prjemian spec2nexus
```

Note that channel *aps-anl-tag* is for production versions while channel *aps-anl-dev* is for development/testing versions. The channel *prjemian* is an alternate with all versions available.

The latest development versions of spec2nexus can be downloaded from the GitHub repository listed above:

```
$ git clone http://github.com/prjemian/spec2nexus.git
```

To install in the standard Python location:

```
$ cd spec2nexus
$ python setup.py install
```

To install in user's home directory:

```
$ python setup.py install --user
```

To install in an alternate location:

```
$ python setup.py install --prefix=/path/to/installation/dir
```

2.1.14 Required Libraries

These libraries are required to write NeXus data files. They are not required to read SPEC data files.

Library	URL
h5py	http://www.h5py.org
numpy	http://numpy.scipy.org/

2.1.15 Optional Libraries

These libraries are used by the *specplot* and *specplot_gallery* modules of the *spec2nexus* package but are not required just to read SPEC data files or write NeXus data files.

Library	URL
MatPlotLib	http://matplotlib.org/

2.1.16 Unit Testing

Since release 2017.0201.0, this project relies on the Python *unittest*[1] package to apply unit testing[2] to the source code. The test code is in the *tests* directory. Various tests have been developed starting with the *2017.0201.0* release to provide features or resolve problems reported. The tests are not yet exhaustive yet the reported code coverage[3] is well over 80%.

The unit tests are implemented in a standard manner such that independent review[4] can run the tests on this code based on the instructions provided in a *.travis.yml* configuration file in the project directory.

This command will run the unit tests locally:

```
python tests
```

Additional information may be learned with a Python package to run the tests:

```
coverage run -a tests && coverage report -m
```

The *coverage* command ([5]), will run the tests and then prepare a report of the percentage of the Python source code that has been executed during the unit tests.

Note: The number of lines reported by *coverage* may differ from that reported by *travis-ci*. The primary reason is that certain tests involving access to information from GitHub may succeed or not depending on the "Github API rate limit".[6]

2.1.17 Example data

About these example data files

These files are examples of various data files that may be read by **spec2nexus**. They are used to test various components of the interface.

[1] Python *unittest* package: https://docs.python.org/2/library/unittest.html

[2] unit testing: https://en.wikipedia.org/wiki/Unit_testing

[3] *coveralls* code coverage: https://coveralls.io/github/prjemian/spec2nexus

[4] *travis-ci* continuous intregration: https://travis-ci.org/prjemian/spec2nexus

[5] *coverage*: https://coverage.readthedocs.io

[6] Github API rate limit: https://developer.github.com/v3/rate_limit/

file		type description
02_03_setup.dat	SPEC scans	1-D scans, some have no data lines (data are stored in HDF5 file)
03_06_JanTest.dat	SPEC scans	1-D scans, USAXS scans, Fly scans, #O+#o and #J+#j control lines
05_02_test.dat	SPEC scans	1-D scans, USAXS scans, Fly scans, multiple #F control lines, multiple #S 1 control lines
33bm_spec.dat	SPEC scans	1-D & 2-D scans (includes hklscan & hklmesh)
33id_spec.dat	SPEC scans	1-D & 2-D scans (includes mesh & Escan scans & MCA data)
APS_spec_data.dat	SPEC scans	1-D scans (ascan & uascan), includes lots of metadata and comments
CdOsO	SPEC scans	1-D scans (ascan), four #E (2, 3659, 3692, 3800) and two #S 1 (35, 3725)
CdSe	SPEC scans	1-D scans (ascan), problem with scan abort on lines 5918-9, in scan 92
compression.h5	NeXus HDF5	2-D compressed image, also demonstrates problem to be resolved in code
Data_Q.h5	NeXus HDF5	2-D image at /entry/data/{I,Q}, test file and variable-length strings
lmn40.spe	SPEC scans	1-D & 2-D scans (hklmesh), two #E lines, has two header sections
mca_spectra_example.dat	SPEC scans	1-D scans (cscan) with 4 MCA spectra in each scan (issue #55)
spec_from_spock.spc	SPEC scans	no header section, uses “nan”, from sardana
startup_1.spec	SPEC scans	1-D scans with SCA spectra & UXML headers for RSM code
user6idd.dat	SPEC scans	1-D scans, aborted scan, control lines: #R #UB #UE #UX #UX1 #UX2 #X, non-default format in #X lines
usaxs-bluesky-specwritercallback.dat	SPEC scans	1-D scans, #MD control lines
writer_1_3.h5	NeXus HDF5	1-D NeXus User Manual example
YSZ011_ALDITO_Fe2O3_planar_fired_1.spc	SPEC scans	1-D scans, text in #V metadata, also has #UIM control lines

Downloads

These downloads are also available online: https://github.com/prjemian/spec2nexus/tree/master/src/spec2nexus/data

- `33bm_spec.dat`
- `33id_spec.dat`
- `APS_spec_data.dat`
- `CdSe`
- `compression.h5`
- `Data_Q.h5`
- `lmn40.spe`

- `mca_spectra_example.dat`
- `user6idd.dat`
- `writer_1_3.h5`
- `YSZ011_ALDITO_Fe2O3_planar_fired_1.spc`

2.1.18 Change History

Production

2021.2.0 release expected 2022-03-15

2021.1.11 released *2022.02.24*

- re-release due to documentation publishing workflow problem

2021.1.10 released *2022.02.24*

- re-release due to documentation publishing workflow problem

2021.1.9 released *2022.02.24*

- **#239** publish documentation at https://prjemian.github.io/spec2nexus/

2021.1.8 released *2020.11.10*

- **#221** move CI from travis-ci to Github Actions, test with python 3.8
- **#217** raise ValueError when `#L` and `#N` lines do not agree

Note: Python 2 end of support

spec2nexus stopped development for Python 2 after release *2021.1.7*, *2019-11-21*. For more information, visit https://python3statement.org/.

2021.1.7 released *2019-11-21*

Note: Last version with support for Python 2

- **#213** copy data file to gallery
- **#208** add more diagnostics to gallery web page comments
- **#191** write each positioner to NXpositioner group
- **#188** catenate continued lines before parsing data
- **#186** remove unused code

2021.1.6 released *2019.11.01*

- **#210** add *-c prjemian* conda channel

2021.1.5 released *2019.11.01*

- **#209** *pyRestTable* added to installation requirements

2021.1.4 released *2019.10.18*

- **#206** specplot_gallery: replot shows all existing plots

2021.1.3 released *2019.08.19* - only update plots with *new* content

- **#202** specplot_gallery: switch to SVG (from PNG) for plots
- **#201** spec: subsequent calls to read() duplicate scans – FIXED
- **#126** spec: new `update_available` property
- **#108** specplot_gallery: only update plots with *new* content

2021.1.2 released *2019.08.15*, plugin enhancements

- **#197** plugins: handle empty empty #O0 or #P0 list
- **#195** drop CII badge: not useful to spec2nexus
- **#190** writer: link content into NXinstrument group
- **#51** plugins: interpret #Gn control lines

2021.1.1 released *2019.07.22*, refactor

- **#181** plugins: revised technique to load control line handlers

2021.1.0 released *2019.07.15*, new features

NEW

- support for `#UXML` metadata
- support for `hklscan` scans
- improved support for `mesh` and `hklmesh` scans
- **#159** handle #UXML metadata control lines
- **#155** module: writer - recognize hklscan
- **#150** module: writer - increase coverage of unit tests: mesh, hklmesh
- **#148** module: eznx - increase coverage of unit tests

2021.0.1 released *2019.07.13*, plugin loading and documentation

- **#170** describe how to write & load Control Line Handler plugins
- **#169** announce deprecation of python 2
- **#165** resolve conda build error
- **#149** unit tests: `units` module

2021.0.0 released *2019.07.12*, API change affecting plugins

API change: Changed how plugins are defined and registered. Custom plugins must be modified and import code revised to work with new system.

- **#168** plugins are now self-registering
- **#166** fix conda packaging

2020.0.2 released *2019.07.09*, bug fixes and code review suggestions

NOTE: conda package is broken (no plugins directory). Only use `pip install spec2nexus` with this release.

- **#164** post conda packages to *aps-anl-tag* channel
- **#161** read files with no #E control line
- **#156** LGTM code review
- **#153** LGTM code review

2020.0.0 released *2019.05.16*, major release

- **#145** unit tests for header content
- **#144** eznx *makeDataset()* now recognizes if data is *ndarray*
- **#123** Accept data files with no header control lines (#F #E #D #C sequence)
- **#113** unit tests for eznx
- **#70** remove h5toText, find this now in *punx* package

2019.0503.0 released *2019.05.03*, tag

- **#142** DuplicateSpecScanNumber with multiple #F sections
- **#137** (again) bug in #U control line handling

2019.0501.0 released *2019.05.01*, tag

- **#137** bug in #U control line handling
- **#140** change: #U data goes into <*object*>.*U* list (name changed from *UserReserved*)

2.1.0 2019.04.26, release

- **#135** switch to semantic versioning
- **#133** support user control line "#U " with plugin
- **#131** support #MD control lines from apstools.SpecWriterCallback
- **#125** fluorescence spectra in files for RSM3D
- **#120** do not mock *six* package in documentation
- **#119** delimiters in #H/#V lines with or without text values
- **#116** process data from spock

  see [release notes](https://github.com/prjemian/spec2nexus/wiki/releasenotes__2-1-0)

  It takes a couple steps to upgrade an existing conda installation from version 2017.nnnn to newer version 2.1.0

  - add a declaration of *spec2nexus < 2000* in the *conda-meta/pinned* file in the conda environment
  - *conda update -c prjemian spec2nexus* (should change to 2.1.0)

  It may still be necessary to uninstall and reinstall spec2nexus to effect an update:

  > conda uninstall -y spec2nexus conda install -c prjemian spec2nexus

2019.0422.0 (tag only)

- tag as-is, for issue #131

2019.0321.0 (tag only)

- tag as-is, post conda noarch package and post to pypi

2017.901.4

- **#62** support Python3
- **#112** merge py3-62 branch
- **#111** Change raise statements to use parens around arguments. Affects issue #62

- **#114** travis-ci for python 3.5 & 3.6
- **#107** Problems accessing SpecDataFileScan.data
- **#95** document final release steps

2017.711.0

- **#110** Ownership of info between #L/data & #S n
- #109 Spaces in data labels on *#L* and other lines

2017.522.1

- #105 ignore extra content in *#@CALIB* control lines
- #104 use versioneer (again)
- **#101** documentation URL & date/time added to every gallery page
- #100 conda package installs properly on Windows now
- #99 BUG: specplot_gallery: plots of hklscan from file *lmn40.spe*
- #98 BUG: specplot_gallery: identify as directory not found
- #52 remove deprecated *prjPySpec* code

2017.317.0

- minor update of the *2017.3.0* release

2017.3.0

- #103 changed *converters* back to *utils*
- #97 PyPI project description now formatted properly
- #90 use *versioneer* (again)

2017-0202.0

- #99 fix list index error in *hklscan* when hkl are all constant
- #96 combine steps when publishing to PyPI

2017-0201.0

- milestone punch list
- #73 refactor mesh and MCA data parsing code
- #67 apply continuous integration via travis-ci
- #66 add verbosity option
- #65 apply unit testing
- #64 *extractSpecScan*: fixed list index out of range
- #63 *extractSpecScan*: command line option to select range of scans
- #56 *specplot* and *specplot_gallery*: add from USAXS instrument and generalize

2016.1025.0 standardize the versioning kit with pyRestTable and pvWebMonitor

2016.1004.0

- #61 release info from git (dropped versioneer package)

2016.0829.0

- #60 Add new plugin test for XPCS plugin (thanks to John Hammonds)

2016.0615.1

- #57 keep information from unrecognized control lines,
- #56 add *specplot* support,
- #55 accept arbitrary number of MCA spectra

2016.0601.0 match complete keys, use unix EOL internally, do not fail if no metadata

2016.0216.0

- #36 identify NIAC2014-compliant NeXus files

2016.0210.0 bugfix: eznx.makeGroup() now correctly sets attributes on new group + documentation for NIAC2014 attributes

2016.0204.0

- #45 handle case when no data points in scan ,
- #46 spec.getScan() ensures argument is used as `str`

2016.0201.0 added spec.getScanNumbersChronological(), spec.getFirstScanNumber(), and spec.getLastScanNumber()

2016.0131.0

- #43 support new NeXus method for default/signal/axes/_indices,

2016.0130.0 fixed #44

2015.1221.1

- #40 added versioneer support

2015.1221.0

- #39 read scans with repeated scan numbers

2015.0822.0 extractSpecScan: add option to report scan heading data, such as positioners and Q

2015.0214.0 h5toText: handle HDF5 'O' data type (variable length strings)

2015.0127.0 spec: ignore bad data lines

2015.0125.0 spec: change handling of #L & #X, refactor detection of scanNum and scanCmd

2015.0113.0 dropped requirement of *lxml* package

2014.1228.1 spec: build mne:name cross-references for counters and positioners

2014.1228.0 show version in documentation

2014.1028.0 spec: quietly ignore unrecognized scan content *for now*

2014.1027.1 spec: major changes in SPEC file support: **custom plugins**

- **spec** based on plugins for each control line, users can add plugins
- declared **prjPySpec** module as legacy, code is frozen at *2014.0623.0* release
- added **spec** module to replace **prjPySpec**

2014.0623.0 updated argparse settings

2014.0622.2 added extractSpecScan.py to the suite from the USAXS project

2014.0410.0 restore scan.fileName variable to keep interface the same for some legacy clients

2014.0404.1 fix sdist utf8 problem, see: http://bugs.python.org/issue11638

2014.0404.0 tree_api_parser moved back into NeXpy project

2014.0320.6 handle multiple header sections in SPEC data file

2014.0320.5 fix the new project URL

2014.0320.4 Sphinx cannot build PDF with code-block in a footnote

2014.0320.3 note the new home URL in the packaging, too, drop nexpy requirement, default docs theme

2014.0320.2 tree_api_parse will go back into nexpy project, remove docs of it here

2014.0320.1 allow readthedocs to build Sphinx without extra package requirements

2014.0320.0

- new home page at http://spec2nexus.readthedocs.org, easier to publish there
- move common methods from __init__.py so docs will build at readthedocs.org
- new test case fails existing SPEC reader, ignore blank lines

2014.03.11 documentation

2014.03.09 h5toText: option to suppress printing of attributes, put URLs in command-line usage documentation, better test of is_spec_file()

2014.03.08 fixed string writer and content display bug in eznx, added h5toText.py, prjPySpec docs improved again

2014.03.051 prjPySpec now handles SPEC v6 data file header additions, add new getScanCommands() method

2014.03.04 (2014_Mardi_Gras release) removed nexpy project requirement from setup, prjPySpec raises exceptions now

2014.03.02 drops nexus tree API (and its dependencies) in favor of native h5py writer

Development: GitHub repository

2014.02.20 version number fits PEP440, LICENSE file included in sdist, more documentation and examples

2014-02-19 reference published documentation (re-posted)

2014-02-19 add documentation framework

2014-02-18 fork to GitHub to make generally available

Development: NeXpy branch

2014-01 briefly, a branch in https://github.com/nexpy/nexpy

- spec2nexus added during this phase
- relies on nexpy.api.nexus for NeXus support

Production: USAXS livedata

2010-2014 production use

- support livedata WWW page of APS USAXS instrument
  - (http://usaxs.xray.aps.anl.gov/livedata/),
- https://subversion.xray.aps.anl.gov/trac/small_angle/browser/USAXS/livedata/prjPySpec.py
- converted from Tcl

2000-2010 Tcl code (*readSpecData.tcl*) in production use at APS sectors 32, 33, & 34

2.1.19 License

```
Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound␣
↪by the terms and conditions of this Creative Commons Attribution 4.0 International␣
↪Public License ("Public License"). To the extent this Public License may be␣
↪interpreted as a contract, You are granted the Licensed Rights in consideration of␣
↪Your acceptance of these terms and conditions, and the Licensor grants You such␣
↪rights in consideration of benefits the Licensor receives from making the Licensed␣
↪Material available under these terms and conditions.

Section 1 -- Definitions.

    Adapted Material means material subject to Copyright and Similar Rights that is␣
↪derived from or based upon the Licensed Material and in which the Licensed Material␣
↪is translated, altered, arranged, transformed, or otherwise modified in a manner␣
↪requiring permission under the Copyright and Similar Rights held by the Licensor.␣
↪For purposes of this Public License, where the Licensed Material is a musical work,␣
↪performance, or sound recording, Adapted Material is always produced where the␣
↪Licensed Material is synched in timed relation with a moving image.
    Adapter's License means the license You apply to Your Copyright and Similar␣
↪Rights in Your contributions to Adapted Material in accordance with the terms and␣
↪conditions of this Public License.
    Copyright and Similar Rights means copyright and/or similar rights closely␣
↪related to copyright including, without limitation, performance, broadcast, sound␣
↪recording, and Sui Generis Database Rights, without regard to how the rights are␣
↪labeled or categorized. For purposes of this Public License, the rights specified␣
↪in Section 2(b)(1)-(2) are not Copyright and Similar Rights.
    Effective Technological Measures means those measures that, in the absence of␣
↪proper authority, may not be circumvented under laws fulfilling obligations under␣
↪Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or␣
↪similar international agreements.
    Exceptions and Limitations means fair use, fair dealing, and/or any other␣
↪exception or limitation to Copyright and Similar Rights that applies to Your use of␣
↪the Licensed Material.
    Licensed Material means the artistic or literary work, database, or other␣
↪material to which the Licensor applied this Public License.
    Licensed Rights means the rights granted to You subject to the terms and␣
↪conditions of this Public License, which are limited to all Copyright and Similar␣
↪Rights that apply to Your use of the Licensed Material and that the Licensor has␣
↪authority to license.
    Licensor means the individual(s) or entity(ies) granting rights under this Public␣
↪License.
```

(continues on next page)

(continued from previous page)

```
    Share means to provide material to the public by any means or process that␣
↪requires permission under the Licensed Rights, such as reproduction, public display,
↪ public performance, distribution, dissemination, communication, or importation,␣
↪and to make material available to the public including in ways that members of the␣
↪public may access the material from a place and at a time individually chosen by␣
↪them.
    Sui Generis Database Rights means rights other than copyright resulting from␣
↪Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on␣
↪the legal protection of databases, as amended and/or succeeded, as well as other␣
↪essentially equivalent rights anywhere in the world.
    You means the individual or entity exercising the Licensed Rights under this␣
↪Public License. Your has a corresponding meaning.

Section 2 -- Scope.

    License grant.
        Subject to the terms and conditions of this Public License, the Licensor␣
↪hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive,␣
↪irrevocable license to exercise the Licensed Rights in the Licensed Material to:
            reproduce and Share the Licensed Material, in whole or in part; and
            produce, reproduce, and Share Adapted Material.
        Exceptions and Limitations. For the avoidance of doubt, where Exceptions and␣
↪Limitations apply to Your use, this Public License does not apply, and You do not␣
↪need to comply with its terms and conditions.
        Term. The term of this Public License is specified in Section 6(a).
        Media and formats; technical modifications allowed. The Licensor authorizes␣
↪You to exercise the Licensed Rights in all media and formats whether now known or␣
↪hereafter created, and to make technical modifications necessary to do so. The␣
↪Licensor waives and/or agrees not to assert any right or authority to forbid You␣
↪from making technical modifications necessary to exercise the Licensed Rights,␣
↪including technical modifications necessary to circumvent Effective Technological␣
↪Measures. For purposes of this Public License, simply making modifications␣
↪authorized by this Section 2(a)(4) never produces Adapted Material.
        Downstream recipients.
            Offer from the Licensor -- Licensed Material. Every recipient of the␣
↪Licensed Material automatically receives an offer from the Licensor to exercise the␣
↪Licensed Rights under the terms and conditions of this Public License.
            No downstream restrictions. You may not offer or impose any additional or␣
↪different terms or conditions on, or apply any Effective Technological Measures to,␣
↪the Licensed Material if doing so restricts exercise of the Licensed Rights by any␣
↪recipient of the Licensed Material.
        No endorsement. Nothing in this Public License constitutes or may be␣
↪construed as permission to assert or imply that You are, or that Your use of the␣
↪Licensed Material is, connected with, or sponsored, endorsed, or granted official␣
↪status by, the Licensor or others designated to receive attribution as provided in␣
↪Section 3(a)(1)(A)(i).

    Other rights.
        Moral rights, such as the right of integrity, are not licensed under this␣
↪Public License, nor are publicity, privacy, and/or other similar personality rights;
↪ however, to the extent possible, the Licensor waives and/or agrees not to assert␣
↪any such rights held by the Licensor to the limited extent necessary to allow You␣
↪to exercise the Licensed Rights, but not otherwise.
        Patent and trademark rights are not licensed under this Public License.
        To the extent possible, the Licensor waives any right to collect royalties␣
↪from You for the exercise of the Licensed Rights, whether directly or through a␣
↪collecting society under any voluntary or waivable statutory or compulsory␣
↪licensing scheme. In all other cases the Licensor expressly reserves any right to␣
↪collect such royalties.
```

(continues on next page)

(continued from previous page)

```
Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following␣
↪conditions.

    Attribution.

        If You Share the Licensed Material (including in modified form), You must:
            retain the following if it is supplied by the Licensor with the Licensed␣
↪Material:
                identification of the creator(s) of the Licensed Material and any␣
↪others designated to receive attribution, in any reasonable manner requested by the␣
↪Licensor (including by pseudonym if designated);
                a copyright notice;
                a notice that refers to this Public License;
                a notice that refers to the disclaimer of warranties;
                a URI or hyperlink to the Licensed Material to the extent reasonably␣
↪practicable;
            indicate if You modified the Licensed Material and retain an indication␣
↪of any previous modifications; and
            indicate the Licensed Material is licensed under this Public License, and␣
↪include the text of, or the URI or hyperlink to, this Public License.
        You may satisfy the conditions in Section 3(a)(1) in any reasonable manner␣
↪based on the medium, means, and context in which You Share the Licensed Material.␣
↪For example, it may be reasonable to satisfy the conditions by providing a URI or␣
↪hyperlink to a resource that includes the required information.
        If requested by the Licensor, You must remove any of the information required␣
↪by Section 3(a)(1)(A) to the extent reasonably practicable.
        If You Share Adapted Material You produce, the Adapter's License You apply␣
↪must not prevent recipients of the Adapted Material from complying with this Public␣
↪License.

Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use␣
↪of the Licensed Material:

    for the avoidance of doubt, Section 2(a)(1) grants You the right to extract,␣
↪reuse, reproduce, and Share all or a substantial portion of the contents of the␣
↪database;
    if You include all or a substantial portion of the database contents in a␣
↪database in which You have Sui Generis Database Rights, then the database in which␣
↪You have Sui Generis Database Rights (but not its individual contents) is Adapted␣
↪Material; and
    You must comply with the conditions in Section 3(a) if You Share all or a␣
↪substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your␣
↪obligations under this Public License where the Licensed Rights include other␣
↪Copyright and Similar Rights.

Section 5 -- Disclaimer of Warranties and Limitation of Liability.

    Unless otherwise separately undertaken by the Licensor, to the extent possible,␣
↪the Licensor offers the Licensed Material as-is and as-available, and makes no␣
↪representations or warranties of any kind concerning the Licensed Material, whether␣
↪express, implied, statutory, or other. This includes, without limitation,␣
↪warranties of title, merchantability, fitness for a particular purpose, non-
↪infringement, absence of latent or other defects, accuracy, or the presence or␣
↪absence of errors, whether or not known or discoverable. Where disclaimers of␣
↪warranties are not allowed in full or in part, this disclaimer may not apply to You.
```

(continues on next page)

(continued from previous page)

```
    To the extent possible, in no event will the Licensor be liable to You on any␣
↪legal theory (including, without limitation, negligence) or otherwise for any␣
↪direct, special, indirect, incidental, consequential, punitive, exemplary, or other␣
↪losses, costs, expenses, or damages arising out of this Public License or use of␣
↪the Licensed Material, even if the Licensor has been advised of the possibility of␣
↪such losses, costs, expenses, or damages. Where a limitation of liability is not␣
↪allowed in full or in part, this limitation may not apply to You.

    The disclaimer of warranties and limitation of liability provided above shall be␣
↪interpreted in a manner that, to the extent possible, most closely approximates an␣
↪absolute disclaimer and waiver of all liability.

Section 6 -- Term and Termination.

    This Public License applies for the term of the Copyright and Similar Rights␣
↪licensed here. However, if You fail to comply with this Public License, then Your␣
↪rights under this Public License terminate automatically.

    Where Your right to use the Licensed Material has terminated under Section 6(a),␣
↪it reinstates:
        automatically as of the date the violation is cured, provided it is cured␣
↪within 30 days of Your discovery of the violation; or
        upon express reinstatement by the Licensor.
    For the avoidance of doubt, this Section 6(b) does not affect any right the␣
↪Licensor may have to seek remedies for Your violations of this Public License.
    For the avoidance of doubt, the Licensor may also offer the Licensed Material␣
↪under separate terms or conditions or stop distributing the Licensed Material at␣
↪any time; however, doing so will not terminate this Public License.
    Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 -- Other Terms and Conditions.

    The Licensor shall not be bound by any additional or different terms or␣
↪conditions communicated by You unless expressly agreed.
    Any arrangements, understandings, or agreements regarding the Licensed Material␣
↪not stated herein are separate from and independent of the terms and conditions of␣
↪this Public License.

Section 8 -- Interpretation.

    For the avoidance of doubt, this Public License does not, and shall not be␣
↪interpreted to, reduce, limit, restrict, or impose conditions on any use of the␣
↪Licensed Material that could lawfully be made without permission under this Public␣
↪License.
    To the extent possible, if any provision of this Public License is deemed␣
↪unenforceable, it shall be automatically reformed to the minimum extent necessary␣
↪to make it enforceable. If the provision cannot be reformed, it shall be severed␣
↪from this Public License without affecting the enforceability of the remaining␣
↪terms and conditions.
    No term or condition of this Public License will be waived and no failure to␣
↪comply consented to unless expressly agreed to by the Licensor.
    Nothing in this Public License constitutes or may be interpreted as a limitation␣
↪upon, or waiver of, any privileges and immunities that apply to the Licensor or You,
↪ including from the legal processes of any jurisdiction or authority.
```

2.2 Indices and tables

- genindex
- modindex
- search

Python Module Index

Index