Astrodata Cheat Sheet

Release 2.1.1

Kathleen Labrie

April 2020

Contents

Document ID

PIPE-USER-105_AstrodataCheatSheet

CHAPTER 1

Astrodata Cheat Sheet

A data package is available for download if you wish to run the examples included in this cheat sheet. Download it at: KL??? URL

To unpack:

```
$ cd <somewhere_convenient>
$ tar xvf ad_usermanual_datapkg-v1.tar
$ bunzip2 ad_usermanual/playdata/*.bz2
```

Then go to the ‘‘ad_usermanual/playground‘ directory to run the examples.

1.1 Imports

Import `astrodata` and `gemini_instruments`:

```
>>> import astrodata
>>> import gemini_instruments
```

1.2 Basic read and write operations

Open a file:

```
>>> ad = astrodata.open('../playdata/N20170609S0154.fits')
```

Get path and filename:

```
>>> ad.path
'../playdata/N20170609S0154.fits'
>>> ad.filename
'N20170609S0154.fits'
```

Write to a new file:

```
>>> ad.write(filename='new154.fits')
>>> ad.filename
N20170609S0154.fits
```

Overwrite the file:

```
>>> adnew = astrodata.open('new154.fits')
>>> adnew.filename
new154.fits
>>> adnew.write(overwrite=True)
```

1.3 Object structure

1.3.1 Description

The `AstroData` object is assigned by "tags" that describe the type of data it contains. The tags are drawn from rules defined in `gemini_instruments` and are based on header information.

When mapping a FITS file, each science pixel extension is loaded as a `NDAstroData` object. The list is zero-indexed. So FITS extension 1 becomes element 0 of the `AstroData` object. If a `VAR` extension is present, it is loaded to the variance attribute of the `NDAstroData`. If a `DQ` extension is present, it is loaded to the `.mask` attribute of the `NDAstroData`. `SCI`, `VAR` and `DQ` are associated through the `EXTVER` keyword value.

In the file below, each `AstroData` "extension" contains the pixel data, then an error plane (`.variance`) and a bad pixel mask plane (`.mask`). `Table` can be attached to an extension, like OBJCAT, or to the `AstroData` object globally, like REFCAT. (In this case, OBJCAT is a catalogue of the sources detected in the image, REFCAT is a reference catalog for the area covered by the whole file.) If other 2D data needs to be associated with an extension this can also be done, like here with OBJMASK, a 2D mask matching the sources in the image.

```
>>> ad = astrodata.open('../playdata/N20170609S0154_varAdded.fits')
>>> ad.info()
Filename: ../playdata/N20170609S0154_varAdded.fits
Tags: ACQUISITION GEMINI GMOS IMAGE NORTH OVERSCAN_SUBTRACTED OVERSCAN_TRIMMED
    PREPARED SIDEREAL
Pixels Extensions
Index  Content                 Type              Dimensions     Format
[ 0]   science                 NDAstroData       (2112, 256)    float32
          .variance            ndarray           (2112, 256)    float32
          .mask                ndarray           (2112, 256)    int16
          .OBJCAT              Table             (6, 43)        n/a
          .OBJMASK             ndarray           (2112, 256)    uint8
[ 1]   science                 NDAstroData       (2112, 256)    float32
          .variance            ndarray           (2112, 256)    float32
          .mask                ndarray           (2112, 256)    int16
          .OBJCAT              Table             (8, 43)        n/a
          .OBJMASK             ndarray           (2112, 256)    uint8
[ 2]   science                 NDAstroData       (2112, 256)    float32
          .variance            ndarray           (2112, 256)    float32
          .mask                ndarray           (2112, 256)    int16
          .OBJCAT              Table             (7, 43)        n/a
          .OBJMASK             ndarray           (2112, 256)    uint8
[ 3]   science                 NDAstroData       (2112, 256)    float32
          .variance            ndarray           (2112, 256)    float32
```

(continues on next page)

(continued from previous page)

```
          .mask                  ndarray           (2112, 256)     int16
          .OBJCAT                Table             (5, 43)         n/a
          .OBJMASK               ndarray           (2112, 256)     uint8
Other Extensions
               Type        Dimensions
.REFCAT        Table       (245, 16)
```

1.3.2 Modifying the structure

Let's first get our play data loaded. You are encouraged to do a `info()` before and after each structure-modification step, to see how things change.

```
>>> from copy import deepcopy
>>> ad = astrodata.open('../playdata/N20170609S0154.fits')
>>> adcopy = deepcopy(ad)
>>> advar = astrodata.open('../playdata/N20170609S0154_varAdded.fits')
```

Append an extension:

```
>>> adcopy.append(advar[3])
>>> adcopy.append(advar[3].data)
```

Delete an extension:

```
>>> del adcopy[5]
```

Delete and add variance and mask planes:

```
>>> var = adcopy[4].variance
>>> adcopy[4].variance = None
>>> adcopy[4].variance = var
```

Attach a table to an extension:

```
>>> adcopy[3].append(advar[0].OBJCAT, name='BOB')
```

Attach a table to the `AstroData` object:

```
>>> adcopy.append(advar.REFCAT, name='BILL')
```

Delete a table:

```
>>> del adcopy[3].BOB
>>> del adcopy.BILL
```

1.4 Astrodata tags

```
>>> ad = astrodata.open('../playdata/N20170521S0925_forStack.fits')
>>> ad.tags
set(['GMOS', 'GEMINI', 'NORTH', 'SIDEREAL', 'OVERSCAN_TRIMMED', 'IMAGE',
'OVERSCAN_SUBTRACTED', 'PREPARED'])
```

(continues on next page)

(continued from previous page)

```
>>> type(ad.tags)
<type 'set'>

>>> {'IMAGE', 'PREPARED'}.issubset(ad.tags)
True
>>> 'PREPARED' in ad.tags
True
```

1.5 Headers

The use of descriptors is favored over direct header access when retrieving values already represented by descriptors, and when writing instrument agnostic routines.

1.5.1 Descriptors

```
>>> ad = astrodata.open('../playdata/N20170609S0154.fits')
>>> ad.filter_name()
'open1-6&g_G0301'
>>> ad.filter_name(pretty=True)
'g'
>>> ad.gain()   # uses a look-up table to get the correct values
[2.03, 1.97, 1.96, 2.01]
>>> ad.hdr['GAIN']
[1.0, 1.0, 1.0, 1.0]    # the wrong values contained in the raw data.
>>> ad[0].gain()
2.03
>>> ad.gain()[0]
2.03

>>> ad.descriptors
('airmass', 'amp_read_area', 'ao_seeing', ...
 ...)
```

1.5.2 Direct access to header keywords

```
>>> ad = astrodata.open('../playdata/N20170609S0154_varAdded.fits')
```

Primary Header Unit

To see a print out of the full PHU:

```
>>> ad.phu
```

Get value from PHU:

```
>>> ad.phu['EXPTIME']
1.0

>>> default = 5.
```

(continues on next page)

(continued from previous page)

```
>>> ad.phu.get('BOGUSKEY', default)
5.0
```

Set PHU keyword, with and without comment:

```
>>> ad.phu['NEWKEY'] = 50.
>>> ad.phu['ANOTHER'] = (30., 'Some comment')
```

Delete PHU keyword:

```
>>> del ad.phu['NEWKEY']
```

Pixel extension header

To see a print out of the full header for an extension or all the extensions:

```
>>> ad[0].hdr
>>> list(ad.hdr)
```

Get value from an extension header:

```
>>> ad[0].hdr['OVERSCAN']
469.7444308769482
>>> ad[0].hdr.get('OVERSCAN', default)
```

Get keyword value for all extensions:

```
>>> ad.hdr['OVERSCAN']
[469.7444308769482, 469.656175780001, 464.9815279808291, 467.5701178951787]
>>> ad.hdr.get('BOGUSKEY', 5.)
[5.0, 5.0, 5.0, 5.0]
```

Set extension header keyword, with and without comment:

```
>>> ad[0].hdr['NEWKEY'] = 50.
>>> ad[0].hdr['ANOTHER'] = (30., 'Some comment')
```

Delete an extension keyword:

```
>>> del ad[0].hdr['NEWKEY']
```

Table header

See the Tables section.

1.6 Pixel data

1.6.1 Arithmetics

Arithmetics with variance and mask propagation is offered for +, −, ⋆, /, and ⋆⋆.

```
>>> ad_hcont = astrodata.open('../playdata/N20170521S0925_forStack.fits')
>>> ad_halpha = astrodata.open('../playdata/N20170521S0926_forStack.fits')

>>> adsub = ad_halpha - ad_hcont

>>> ad_halpha[0].data.mean()
646.11896
>>> ad_hcont[0].data.mean()
581.81342
>>> adsub[0].data.mean()
64.305862

>>> ad_halpha[0].variance.mean()
669.80664
>>> ad_hcont[0].variance.mean()
598.46667
>>> adsub[0].variance.mean()
1268.274


# In place multiplication
>>> ad_mult = deepcopy(ad)
>>> ad_mult.multiply(ad)
>>> ad_mult.multiply(5.)


# Using descriptors to operate in-place on extensions.
>>> from copy import deepcopy
>>> ad = astrodata.open('../playdata/N20170609S0154_varAdded.fits')
>>> ad_gain = deepcopy(ad)
>>> for (ext, gain) in zip(ad_gain, ad_gain.gain()):
...     ext.multiply(gain)
>>> ad_gain[0].data.mean()
366.39545
>>> ad[0].data.mean()
180.4904
>>> ad[0].gain()
2.03
```

1.6.2 Other pixel data operations

```
>>> import numpy as np
>>> ad_halpha[0].mask[300:350,300:350] = 1
>>> np.mean(ad_halpha[0].data[ad_halpha[0].mask==0])
657.1994
>>> np.mean(ad_halpha[0].data)
646.11896
```

1.7 Tables

Tables are stored as `astropy.table.Table` class. FITS tables are represented in `astrodata` as `Table` and FITS headers are stored in the `NDAstroData` `meta` attribute. Most table access should be done through the `Table` interface. The best reference is the `astropy` documentation itself. Below are just a few examples.

```
>>> ad = astrodata.open('../playdata/N20170609S0154_varAdded.fits')
```

Get column names:

```
>>> ad.REFCAT.colnames
```

Get column content:

```
>>> ad.REFCAT['zmag']
>>> ad.REFCAT['zmag', 'zmag_err']
```

Get content of row:

```
>>> ad.REFCAT[4]     # 5th row
>>> ad.REFCAT[4:6]   # 5th and 6th rows
```

Get content from specific row and column:

```
>>> ad.REFCAT['zmag'][4]
```

Add a column:

```
>>> new_column = [0] * len(ad.REFCAT)
>>> ad.REFCAT['new_column'] = new_column
```

Add a row:

```
>>> new_row = [0] * len(ad.REFCAT.colnames)
>>> ad.REFCAT.add_row(new_row)
```

Selecting value from criterion:

```
>>> ad.REFCAT['zmag'][ad.REFCAT['Cat_Id'] == '1237662500002005475']
>>> ad.REFCAT['zmag'][ad.REFCAT['zmag'] < 18.]
```

Rejecting `numpy.nan` before doing something with the values:

```
>>> t = ad.REFCAT   # to save typing.
>>> t['zmag'][np.where(np.isnan(t['zmag']), 99, t['zmag']) < 18.]

>>> t['zmag'].mean()
nan
>>> t['zmag'][np.where(~np.isnan(t['zmag']))].mean()
20.2924
```

If for some reason you need to access the FITS table headers, here is how to do it.

To see the FITS headers:

```
>>> ad.REFCAT.meta
>>> ad[0].OBJCAT.meta
```

To retrieve a specific FITS table header:

```
>>> ad.REFCAT.meta['header']['TTYPE3']
'RAJ2000'
>>> ad[0].OBJCAT.meta['header']['TTYPE3']
'Y_IMAGE'
```

To retrieve all the keyword names matching a selection:

```
>>> keynames = [key for key in ad.REFCAT.meta['header'] if key.startswith('TTYPE')]
```

1.8 Create new AstroData object

Basic header and data array set to zeros:

```
>>> from astropy.io import fits

>>> phu = fits.PrimaryHDU()
>>> pixel_data = np.zeros((100,100))

>>> hdu = fits.ImageHDU()
>>> hdu.data = pixel_data
>>> ad = astrodata.create(phu)
>>> ad.append(hdu, name='SCI')

or another way:
>>> hdu = fits.ImageHDU(data=pixel_data, name='SCI')
>>> ad = astrodata.create(phu, [hdu])
```

A `Table` as an `AstroData` object:

```
>>> from astropy.table import Table

>>> my_astropy_table = Table(list(np.random.rand(2,100)), names=['col1', 'col2'])
>>> phu = fits.PrimaryHDU()

>>> astrodata.add_header_to_table(my_astropy_table)
>>> ad = astrodata.create(phu)
>>> ad.append(my_astropy_table, name='BOB')

From a BinTableHDU:
>>> phu = fits.PrimaryHDU()
>>> ad = astrodata.create(phu)
>>> ad.append(my_fits_table, name='BOB')

WARNING: This last line will not run like the others as we have not defined
"my_fits_table".  This is nonetheless how it is done if you had a FITS table.
```