AstroAsciiData

User Manual version 1.1.1

M. Kümmel (mkuemmel@eso.org), J. Haase (jhaase@eso.org)
Space Telescope - European Coordinating Facility

28 March 2009

Contents

1 Introduction

1.1 ASCII tables in astronomy and science

ASCII tables are one of the major data exchange formats used in science. In astronomy, which is the background of `AstroAsciiData` 's developers, ASCII tables are used for a variety of things like object lists, line lists or even spectra. Every person working in astronomy has to deal with ASCII data, and there are various ways of doing so. Some use the `awk` scripting language, some transfer the ASCII tables to FITS tables and then work on the FITS data, some use IDL routines. Most of those approaches need individual efforts (such as preparing a format file for the transformation to FITS) whenever there is a new kind of ASCII table with e.g. a different number of columns.

1.2 The project goal

Within the `AstroAsciiData` project we envision a module which can be used to work on **all** kinds of ASCII tables. The module provides a convenient tool such that the user easily can:

- read in ASCII tables;
- manipulate table elements;
- save the modified ASCII table;
- read and write meta data such as column names and units;
- combine several tables;
- delete/add rows and columns;
- manage metadata in the table headers.

1.3 Why python?

Python (www.python.org) is in the process of becomming the programming language of choice for astronomers and scientists in general, both for interactive data analysis as well as for large scale software development. A number of interfaces such as PyRAF (http://www.stsci.edu/resources/software_hardware/pyraf) or PyFITS (http://www.stsci.edu/resources/software_hardware/pyfits) have already been written to bridge the gap between widely used astronomical software packages, data formats and Python.

This makes the development of the `AstroAsciiData` module for Python a natural choice. Within Python, the `AstroAsciiData` module may be used interactively, within small scripts, in data reduction tasks and even in data bases.

1.4 Design considerations

In general, the ASCII tables used in astronomy have a relatively small size. As an example, the size of the Wide Field Camera catalogue of Hubble Ultra Deep Field (http://www.stsci.edu/hst/udf) is only 2.2 MB. Handling those amounts of data is not a time consuming task for modern day computers. As a consequence, computational speed is not a prime issue in software design and construction, and there was no attempt to implement `AstroAsciiData` as a particularly fast module. The focus was rather to maximizing convenience and ensuring a steep learning curve for the users.

1.5 The SExtractor table format

There are many ways to store meta data such column name and units in a file together with the table data. Instead of defining our own, proprietary format within the `AstroAsciiData` module, we have chosen to support the SExtractor header scheme.

This means that the module can read ASCII tables which follow the SExtractor format and extract all column information from the file (see Sect. 3.3). The module also offers to write this information in the SExtractor format back into file.

In the SExtractor format the meta data is stored at the beginning of the file:

```
#   1 NUMBER          Running object number
#   2 XWIN_IMAGE      Windowed position estimate along x             [pixel]
#   3 YWIN_IMAGE      Windowed position estimate along y             [pixel]
#   4 ERRY2WIN_IMAGE  Variance of windowed pos along y               [pixel**2]
#   5 AWIN_IMAGE      Windowed profile RMS along major axis          [pixel]
#   6 ERRAWIN_IMAGE   RMS windowed pos error along major axis        [pixel]
#   7 BWIN_IMAGE      Windowed profile RMS along minor axis          [pixel]
#   8 ERRBWIN_IMAGE   RMS windowed pos error along minor axis        [pixel]
#   9 MAG_AUTO        Kron-like elliptical aperture magnitude        [mag]
#  10 MAGERR_AUTO     RMS error for AUTO magnitude                   [mag]
#  11 CLASS_STAR      S/G classifier output
#  12 FLAGS           Extraction flags
  1 100.523  11.911   2.783 0.0693 2.078 0.0688  -5.3246   0.0416  0.00  19
  2 100.660   4.872   7.005 0.1261 3.742 0.0989  -6.4538   0.0214  0.00  27
  3 131.046  10.382   1.965 0.0681 1.714 0.0663  -4.6836   0.0524  0.00  17
  4 338.959   4.966  11.439 0.1704 4.337 0.1450  -7.1747   0.0173  0.00  25
```

The format is rather simple, but nevertheless offers the possibility to save the essential column information.

Potential users who would need or prefer other formats can:

- try to convince us that the alternative format is worth the implementation;
- sub-class the relevant module classes and implement the format support by themselves. We would certainly offer help for this.

1.6 The usage of the User Manual

With more than 70 pages this User Manual is quite voluminous. But to start reading at page 1 and working through it all is pointless. Our recommendations on how to proceed are the following:

- install the module with the the help of Sect. 2;
- get a first impression on `AstroAsciiData` browsing through Sect. 3;
- start working with the module;
- use Sect. 4 with its detailed examples as a reference manual, preferably in the `html` form (available at the project site http://www.stecf.org/software/PYTHONtools/astroasciidata/) when looking for a certain functionality or features you need.

1.7 Feedback

Feedback in any form, suggestions, critics, comments, development requests, is very much welcome and will certainly contribute to improve the next versions of the module. The feedback should be sent directly to the developers or to `AstroAsciiData@stecf.org`.

2 Installation

The AstroAsciiData module requires Python 2.4 or later. It was developed on linux (SUSE, fedora), Solaris 5.8 and MacOSX, however there should be no problems installing it on any machine hosting Python.

Individual functions, such as the transformation to numarray, numpy or the FITS format, obviously require the python modules for the formats they convert to. But there is no general need to install them.

The current version 1.1 of AstroAsciiData is distributed as the source archive asciidata-1.1.tar.gz from the AstroAsciiData webpage at http://www.stecf.org/software/PYTHONtools/astroasciidata/. Installing the module is not difficult. Unpack the tarball with:

```
> gunzip asciidata-1.1.tar.gz
> tar -xvf asciidata-1.1.tar
```

Then enter the the unpacked directory and do the usual:

```
> cd asciidata-1.1
> python setup.py install
```

After installation, some Unit Tests can be executed with:

```
> python setup.py test
```

If there are no errors reported in the Unit Tests, the proper working of the module is assured. Failed tests may happen due to missing pyhon modules (numpy, PyFITS or numarray) and can be neglected if you do not intend to convert into these formats.

In all classes and sub-modules the epydoc-conventions have been used in the inline documentation. In case that epydoc (http://epydoc.sourceforge.net/) is installed, the command

```
> epydoc Lib/
```

creates webpages from the inline documenatation, which are written to the the directory './html'. This would be certainly a very good start for users who really want to find out what is behind the module or intend to subclass it to e.g. support their own, custom made ASCII table format with column names. In case that you just want to use the AstroAsciiData module, there is no need to look at its inline documentation.

2.1 Lemma for version 1.1.1

This newest version is rather a patch release. The advent of numpy-1.1 require some small changes not even to the code itself, but to the unit tests. Since these are integral part of the software release, we decided to upgrade and call the new bundle version 1.1.1.

2.2 Release notes for version 1.1

Except for minor bug fixes, version 1.1 contains the following improvements:

- export to numpy;
- export to FITS via numpy;
- some convenient functions were added (see e.g. Sect.4.2.15);
- column names can contain basic arithmetic operators now.
- code with deprecation warnings was replaced.

3 All in one chapter

Reading documentation is no fun. Moreover the `AstroAsciiData` module promised to be convenient for users (see Sect. 1.2). This Section gives a fast introduction on all features of the `AstroAsciiData` module and how these features are used to work with ASCII tables. On the basis of some sample session the most important classes and methods are introduced without explicitly listing all their names and modes of usage. A complete and detailed overview on all `AstroAsciiData` classes and their methods is given in Section 4.

3.1 Working with existing data

This chapter shows how to load and work with the ASCII table 'example.txt'. This tables looks like:

```
#
# Some objects in the GOODS field
#
unknown 189.2207323  62.2357983 26.87 0.32
galaxy  189.1408929  62.2376331 24.97 0.15
star    189.1409453  62.1696844 25.30 0.12
galaxy  188.9014716  62.2037839 25.95 0.20
```

⇒ Before actually loading the table, the `AstroAsciiData` module must be imported with:

```
>>> import asciidata
```

⇒ The ASCII table is loaded with:

```
>>> example = asciidata.open('example.txt')
```

⇒ Just to check whether the table was loaded correctly you do:

```
>>> print str(example)
#
# Some objects in the GOODS field
#
unknown  189.2207323  62.2357983  26.87  0.32
 galaxy  189.1408929  62.2376331  24.97  0.15
   star  189.1409453  62.1696844  25.30  0.12
 galaxy  188.9014716  62.2037839  25.95  0.20
```

⇒ As a first application, you want to compute the average from the numbers in the second and third row:

```
>>> sum1=0.0
>>> sum2=0.0
>>> for index in range(example.nrows):
...     sum1 += example[1][index]
...     sum2 += example[2][index]
...
>>> ave1 = sum1/example.nrows
>>> ave2 = sum2/example.nrows
>>> print ave1, ave2
189.101010525 62.211724925
```

Please note that indices start with 0, so the first row in the first column is `example[0][0]`.

⇒ You want to change the table values, but before that perhaps it would be wise to keep a copy of the original ASCII table :

```
>>> example.writeto('example_orig.txt')
```

This gives you a file 'example_orig.txt', which is identical to the original 'example.txt'.

⇒ Now you may want to compute and save the differences between the average and the individual values:

```
>>> for index in range(example.nrows):
...     example['diff1'][index] = example[1][index] - ave1
...     example['diff2'][index] = example[2][index] - ave2
...
>>> print str(example)
#
# Some objects in the GOODS field
#
unknown  189.2207323  62.2357983  26.87  0.32  1.197218e-01  2.407338e-02
 galaxy  189.1408929  62.2376331  24.97  0.15  3.988237e-02  2.590817e-02
   star  189.1409453  62.1696844  25.30  0.12  3.993477e-02 -4.204052e-02
 galaxy  188.9014716  62.2037839  25.95  0.20 -1.995389e-01 -7.941025e-03
```

There are two new columns, which were created by addressing elements in an unknown column with the name 'diff1' and 'diff2'.

⇒ To remember the new columns and their meaning, you would like to put a note into the table header:

```
>>> example.header.append('Nov 16 2005: computed and stored differences!')
>>> print str(example)
#
# Some objects in the GOODS field
#
```

```
# Nov 16 2005: computed and stored differences!
unknown  189.2207323  62.2357983  26.87  0.32  1.197218e-01  2.407338e-02
 galaxy  189.1408929  62.2376331  24.97  0.15  3.988237e-02  2.590817e-02
   star  189.1409453  62.1696844  25.30  0.12  3.993477e-02 -4.204052e-02
 galaxy  188.9014716  62.2037839  25.95  0.20 -1.995389e-01 -7.941025e-03
```

There is a new commented line at the beginning of the table with your note.

⇒ That was enough for now, and the best is to save the modified ASCII table:

```
>>> example.flush()
```

Now the file 'example.txt' also has the two new columns.

⇒ OK, there is a column with the name 'diff1' and another named 'diff2', but what are the names of the original columns? To get all information, just type:

```
>>> print example.info()
File:        example.txt
Ncols:       7
Nrows:       4
Delimiter:   None
Null value: ['Null', 'NULL', 'None', '*']
Comment:     #
Column name:        column1
Column type:        <type 'str'>
Column format:      ['% 7s', '%7s']
Column null value : ['Null']
Column name:        column2
Column type:        <type 'float'>
Column format:      ['% 11.7f', '%12s']
Column null value : ['Null']
Column name:        column3
Column type:        <type 'float'>
Column format:      ['% 10.7f', '%11s']
Column null value : ['Null']
Column name:        column4
Column type:        <type 'float'>
Column format:      ['% 5.2f', '%6s']
Column null value : ['Null']
Column name:        column5
Column type:        <type 'float'>
Column format:      ['% 4.2f', '%5s']
Column null value : ['Null']
Column name:        diff1
Column type:        <type 'float'>
Column format:      ['% 12.6e', '%13s']
```

```
Column null value : ['Null']
Column name:        diff2
Column type:        <type 'float'>
Column format:      ['% 12.6e', '%13s']
Column null value : ['Null']
```

So the original columns had default names such as 'column1', 'column2', ... Moreover the method `info()` returns the column type and format for every column .

3.2 Creating an ASCII table from scratch

In this Section an ASCII table is created from scratch using functions classes and methods in the `AstroAsciiData` module.

⇒ To create an empty `AsciiData` object, import the `AstroAsciiData` module and type:

```
>>> import asciidata
>>> example2 = asciidata.create(4,10)
>>> print example2
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
      Null       Null       Null       Null
```

The object has four columns with ten rows, all empty so far.

⇒ The first column should contain an index:

```
>>> for index in range(example2.nrows):
...     example2[0][index] = index+1
...
>>> print example2
    1       Null       Null       Null
    2       Null       Null       Null
    3       Null       Null       Null
    4       Null       Null       Null
    5       Null       Null       Null
    6       Null       Null       Null
    7       Null       Null       Null
    8       Null       Null       Null
    9       Null       Null       Null
   10       Null       Null       Null
```

For convenience the index starts with 1.

⇒ Now the rest is filled using a functional form:

```
>>> for cindex in range(1, example2.ncols):
...   for rindex in range(example2.nrows):
...     example2[cindex][rindex] = 0.3*float(example2[0][rindex])**cindex
...
>>> print example2
    1  3.000000e-01  3.000000e-01  3.000000e-01
    2  6.000000e-01  1.200000e+00  2.400000e+00
    3  9.000000e-01  2.700000e+00  8.100000e+00
    4  1.200000e+00  4.800000e+00  1.920000e+01
    5  1.500000e+00  7.500000e+00  3.750000e+01
    6  1.800000e+00  1.080000e+01  6.480000e+01
    7  2.100000e+00  1.470000e+01  1.029000e+02
    8  2.400000e+00  1.920000e+01  1.536000e+02
    9  2.700000e+00  2.430000e+01  2.187000e+02
   10  3.000000e+00  3.000000e+01  3.000000e+02
```

Please note that the column type of the first column is `int` since only integer type data was entered. In all other columns numbers of type float were entered, hence their column type is also `float`.

⇒ Inserting one element with a more generic data type changes the type of the whole column:

```
>>> example2[0][1] = 2.0
>>> print example2
 1.000000e+00  3.000000e-01  3.000000e-01  3.000000e-01
 2.000000e+00  6.000000e-01  1.200000e+00  2.400000e+00
 3.000000e+00  9.000000e-01  2.700000e+00  8.100000e+00
 4.000000e+00  1.200000e+00  4.800000e+00  1.920000e+01
 5.000000e+00  1.500000e+00  7.500000e+00  3.750000e+01
 6.000000e+00  1.800000e+00  1.080000e+01  6.480000e+01
 7.000000e+00  2.100000e+00  1.470000e+01  1.029000e+02
 8.000000e+00  2.400000e+00  1.920000e+01  1.536000e+02
 9.000000e+00  2.700000e+00  2.430000e+01  2.187000e+02
 1.000000e+01  3.000000e+00  3.000000e+01  3.000000e+02
```

Now the first column is of type `float` as well.

⇒ Eventually, some comments are added for specific rows:

```
>>> newcol = example2.append('comment')
>>> example2[newcol][0] =  'small!'
>>> example2[newcol][2] =  'bigger!'
>>> example2[newcol][7] =  'Huge!'
>>> print example2
 1.000000e+00  3.000000e-01  3.000000e-01  3.000000e-01 small!
```

```
 2.000000e+00  6.000000e-01  1.200000e+00  2.400000e+00   Null
 3.000000e+00  9.000000e-01  2.700000e+00  8.100000e+00 bigger!
 4.000000e+00  1.200000e+00  4.800000e+00  1.920000e+01   Null
 5.000000e+00  1.500000e+00  7.500000e+00  3.750000e+01   Null
 6.000000e+00  1.800000e+00  1.080000e+01  6.480000e+01   Null
 7.000000e+00  2.100000e+00  1.470000e+01  1.029000e+02   Null
 8.000000e+00  2.400000e+00  1.920000e+01  1.536000e+02  Huge!
 9.000000e+00  2.700000e+00  2.430000e+01  2.187000e+02   Null
 1.000000e+01  3.000000e+00  3.000000e+01  3.000000e+02   Null
```

Obviously it also is possible to create a new column using the `AsciiData` `append()` method. This method returns the column number, which then can be used to fill the new column.

⇒ Now its time to safe the `AsciiData` object:

```
>>> example2.flush()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File ".../site-packages/asciidata/asciidata.py", line 279, in flush
    raise "No filename given. Use 'writeto()' instead."
No filename given. Use 'writeto()' instead.
>>> example2.writeto('example2.txt')
```

Since the object was created from scratch, there is no filename associated with and the method `flush()` can not be used. The method `writeto()` must be used instead!

3.3 Woorking with SExtractor formatted data

This chapter shows how to load and work with the table 'SExample.txt', which was produced by SExtractor. This object catalogue looks like:

```
#   1 NUMBER          Running object number
#   2 MAG_APER        Fixed aperture magnitude vector                  [mag]
#   5 MAGERR_APER     RMS error vector for fixed aperture mag.         [mag]
#   8 FLUX_AUTO       Flux within a Kron-like elliptical aperture      [count]
#   9 FLUXERR_AUTO    RMS error for AUTO flux                          [count]
#  10 X_IMAGE         Object position along x                          [pixel]
#  11 Y_IMAGE         Object position along y                          [pixel]
#  12 FLAGS           Extraction flags
 1  -7.1135  -9.9589 -11.4873 0.1151 0.0168 0.0082 52533.1 580.708 379.715  72.461 3
 2  -7.8412  -9.5452 -10.8191 0.0591 0.0246 0.0152  171543 2014.45 341.365 320.621 9
 3  -8.1548  -9.6216 -11.0307 0.0444 0.0229 0.0125  267764 1844.97 379.148 196.397 3
 4  -9.4534 -11.0534 -11.9600 0.0134 0.0061 0.0053  178541 1290.41 367.213 123.803 3
 5  -7.8104  -9.1967 -10.5169 0.0609 0.0339 0.0201  131343 1648.34 305.545 307.027 3
 6 -12.1666 -13.2193 -14.1293 0.0011 0.0008 0.0007 2.22738 1938.01 258.692 260.341 3
 7  -8.8319 -10.3340 -11.3343 0.0238 0.0119 0.0095  111597 1525.78 336.462  97.060 3
 8  -8.9203 -10.3532 -11.7252 0.0219 0.0117 0.0066  129934 917.641 177.377 199.843 3
```

```
 9  -7.5366  -9.0374 -10.3321 0.0784 0.0393 0.0238 72761.7 1603.15  94.196 131.380 3
10  -6.7963  -8.4304  -9.5685 0.1552 0.0687 0.0482   14072 895.465 265.404  46.241 3
```

Please note the the jump in the column numbers from *#2 MAG_APER* to *#5 MAGERR_APER* and then *#8 FLUX_AUTO*. *MAG_APER* and *MAGERR_APER* are both vector data with three items each. There are three colmns with *MAG_APER*-values and three columns with *MAGERR_APER*-values in the table data, however the header contains only one explicit entry for *MAG_APER* and *MAGERR_APER*.

In `AstroAsciiData` an individual column name is given to each of these multiple columns by adding a number to the basic name given in the header.

⇒ First you load the `AstroAsciiData` module and the table:

```
>>> import asciidata
>>> SExample = asciidata.open('SExample.txt')
```

⇒ Now you check the columns in the `AsciiData` object

```
>>> print SExample.info()
File:       SExample.cat
Ncols:      12
Nrows:      10
Delimiter:  None
Null value: ['Null', 'NULL', 'None', '*']
comment_char:    #
Column name:        NUMBER
Column type:        <type 'int'>
Column format:      ['%5i', '%5s']
Column null value : ['Null']
Column comment : Running object number
Column name:        MAG_APER
Column type:        <type 'float'>
Column format:      ['% 6.4f', '%7s']
Column null value : ['Null']
Column unit : mag
Column comment : Fixed aperture magnitude vector
Column name:        MAG_APER1
Column type:        <type 'float'>
Column format:      ['% 6.4f', '%7s']
Column null value : ['Null']
Column name:        MAG_APER2
Column type:        <type 'float'>
Column format:      ['% 7.4f', '%8s']
Column null value : ['Null']
Column name:        MAGERR_APER
Column type:        <type 'float'>
Column format:      ['% 6.4f', '%7s']
Column null value : ['Null']
```

```
Column unit : mag
Column comment : RMS error vector for fixed aperture mag.
Column name:        MAGERR_APER1
Column type:        <type 'float'>
Column format:      ['% 6.4f', '%7s']
Column null value : ['Null']
Column name:        MAGERR_APER2
Column type:        <type 'float'>
Column format:      ['% 6.4f', '%7s']
Column null value : ['Null']
Column name:        FLUX_AUTO
Column type:        <type 'float'>
Column format:      ['% 7.1f', '%8s']
Column null value : ['Null']
Column unit : count
Column comment : Flux within a Kron-like elliptical aperture
Column name:        FLUXERR_AUTO
Column type:        <type 'float'>
Column format:      ['% 7.3f', '%8s']
Column null value : ['Null']
Column unit : count
Column comment : RMS error for AUTO flux
Column name:        X_IMAGE
Column type:        <type 'float'>
Column format:      ['% 7.3f', '%8s']
Column null value : ['Null']
Column unit : pixel
Column comment : Object position along x
Column name:        Y_IMAGE
Column type:        <type 'float'>
Column format:      ['% 6.3f', '%7s']
Column null value : ['Null']
Column unit : pixel
Column comment : Object position along y
Column name:        FLAGS
Column type:        <type 'int'>
Column format:      ['%5i', '%5s']
Column null value : ['Null']
Column comment : Extraction flags
```

In this list there are the expanded column names *MAG_APER*, *MAG_APER1* and *MAG_APER2*, and now every data column has a proper name.

⇒ You compute the signa-to-noise-ratio, set the column comment and check it:

```
>>> for ii in range(SExample.nrows):
...     SExample['SNR'][ii]=SExample['FLUX_AUTO'][ii]/SExample['FLUXERR_AUTO'][ii]
...
>>> SExample['SNR'].set_colcomment('Singal-to-Noise-Ratio')
```

```
>>> print SExample['SNR'].info()
Column name:        SNR
Column type:        <type 'float'>
Column format:      ['% 12.6e', '%13s']
Column null value : ['Null']
Column comment : Singal-to-Noise-Ratio
```

⇒ The object is sorted according to the signal-to-noise-ratio, the result is checked and the `AsciiData` object re-written to the disk:

```
>>> SExample.sort('SNR', descending=1)
>>> print SExample['SNR']
Column: SNR
 1.451319e+02
 1.415957e+02
 1.383599e+02
 9.046388e+01
 8.515625e+01
 7.968198e+01
 7.314095e+01
 4.538671e+01
 1.571474e+01
 1.149313e-03
>>> SExample.flush()
```

⇒ Your favoured plotting program can not deal with any kind of header. You tranfer the `AsciiData` object to plain format and write it to a special plotting file 'SExample.plot':

```
>>> SExample.toplain()
>>> SExample.writeto('SExample.plot')
>>>
```

⇒ You are finshed now, leave python and, since you do not trust the `AstroAsciiData` module, check both files:

```
~>more SExample.cat
# 1  NUMBER  Running object number
# 2  MAG_APER  Fixed aperture magnitude vector  [mag]
# 3  MAG_APER1
# 4  MAG_APER2
# 5  MAGERR_APER  RMS error vector for fixed aperture mag.  [mag]
# 6  MAGERR_APER1
# 7  MAGERR_APER2
# 8  FLUX_AUTO  Flux within a Kron-like elliptical aperture  [count]
# 9  FLUXERR_AUTO  RMS error for AUTO flux  [count]
# 10 X_IMAGE  Object position along x  [pixel]
# 11 Y_IMAGE  Object position along y  [pixel]
# 12 FLAGS  Extraction flags
# 13 SNR  Singal-to-Noise-Ratio
    3 -8.1548 -9.6216 -11.0307  0.0444  0.0229  0.0125  267764.0  1844.970  379.148  196.397     3  1.451319e+02
    8 -8.9203 -10.3532 -11.7252  0.0219  0.0117  0.0066  129934.0  917.641  177.377  199.843     3  1.415957e+02
    4 -9.4534 -11.0534 -11.9600  0.0134  0.0061  0.0053  178541.0  1290.410  367.213  123.803     3  1.383599e+02
    1 -7.1135 -9.9589 -11.4873  0.1151  0.0168  0.0082  52533.1  580.708  379.715  72.461     3  9.046388e+01
    2 -7.8412 -9.5452 -10.8191  0.0591  0.0246  0.0152  171543.0  2014.450  341.365  320.621     9  8.515625e+01
    5 -7.8104 -9.1967 -10.5169  0.0609  0.0339  0.0201  131343.0  1648.340  305.545  307.027     3  7.968198e+01
    7 -8.8319 -10.3340 -11.3343  0.0238  0.0119  0.0095  111597.0  1525.780  336.462  97.060     3  7.314095e+01
    9 -7.5366 -9.0374 -10.3321  0.0784  0.0393  0.0238  72761.7  1603.150  94.196  131.380     3  4.538671e+01
   10 -6.7963 -8.4304 -9.5685  0.1552  0.0687  0.0482  14072.0  895.465  265.404  46.241     3  1.571474e+01
    6 -12.1666 -13.2193 -14.1293  0.0011  0.0008  0.0007     2.2  1938.010  258.692  260.341     3  1.149313e-03
~>
~>more SExample.plot
```

```
    3 -8.1548 -9.6216 -11.0307  0.0444  0.0229  0.0125  267764.0  1844.970  379.148  196.397     3  1.451319e+02
    8 -8.9203 -10.3532 -11.7252  0.0219  0.0117  0.0066  129934.0  917.641  177.377  199.843     3  1.415957e+02
    4 -9.4534 -11.0534 -11.9600  0.0134  0.0061  0.0053  178541.0  1290.410  367.213  123.803     3  1.383599e+02
    1 -7.1135 -9.9589 -11.4873  0.1151  0.0168  0.0082  52533.1  580.708  379.715  72.461     3  9.046388e+01
    2 -7.8412 -9.5452 -10.8191  0.0591  0.0246  0.0152  171543.0  2014.450  341.365  320.621     9  8.515625e+01
    5 -7.8104 -9.1967 -10.5169  0.0609  0.0339  0.0201  131343.0  1648.340  305.545  307.027     3  7.968198e+01
    7 -8.8319 -10.3340 -11.3343  0.0238  0.0119  0.0095  111597.0  1525.780  336.462  97.060     3  7.314095e+01
    9 -7.5366 -9.0374 -10.3321  0.0784  0.0393  0.0238  72761.7  1603.150  94.196  131.380     3  4.538671e+01
   10 -6.7963 -8.4304 -9.5685  0.1552  0.0687  0.0482  14072.0  895.465  265.404  46.241     3  1.571474e+01
    6 -12.1666 -13.2193 -14.1293  0.0011  0.0008  0.0007     2.2  1938.010  258.692  260.341     3  1.149313e-03
~>
```

The two files contain the same data. In the SExtractor version the column names are still present in the header.

4 The detailed description

The `AstroAsciiData` module was developed in Python using an Object Oriented (OO) approach with classes and methods. This can not be hidden in the usage of the `AstroAsciiData` module. Working with `AstroAsciiData` means creating its class objects, accessing the class data and executing class methods. This might be confusing for users who are not familiar with this terminology and its meaning.

However this manual makes no attempt to introduce the OO terminology, and its complete understanding is not really necessary in order to use the `AstroAsciiData` module . The user can simply stick to a strictly *phenomenological* approach by looking at the examples and transferring them to his/her own applications. Nevertheless the OO terms are used to structure this section of the manual.

4.1 Functions

The `AstroAsciiData` module contains the two functions `open()` and `create()`. These function serve as a starting point for the work with ASCII tables, since both return an `AsciiData` object by either opening and loading an existing ASCII file (`open()`) or creating an empty `AsciiData` object from scratch (`create()`).

4.1.1 open()

This function loads an existing ASCII table file. An `AsciiData` object is created and the data stored in the ASCII table is transferred to the AsciiData object. Various function parameters specify e.g. the character used as a delimiter to separate adjacent column elements.

Usage
open(filename, null=None, delimiter=None, comment_char=None)

Parameters

Name	Type	Default	Description
filename	*string*	-	the name of the ASCII table file to be loaded
null	*string*	['*','NULL', 'Null', 'None']	the character/string representing a null-entry
delimiter	*string*	" "	the delimiter separating columns
comment_char	*string*	'#'	the character/string indicating a comment

Return
- an `AsciiData` object

Examples

1. Load the file 'example.txt' and print the result. The file 'example.txt looks like:

```
#
# Some objects in the GOODS field
#
unknown  189.2207323  62.2357983  26.87  0.32
 galaxy  189.1408929  62.2376331  24.97  0.15
   star  189.1409453  62.1696844  25.30  0.12
 galaxy  188.9014716  62.2037839  25.95  0.20
```

   The command sequence is:

```
>>> example = asciidata.open('example.txt')
>>> print example
#
# Some objects in the GOODS field
#
unknown  189.2207323  62.2357983  26.87  0.32
 galaxy  189.1408929  62.2376331  24.97  0.15
   star  189.1409453  62.1696844  25.30  0.12
 galaxy  188.9014716  62.2037839  25.95  0.20
```

2. Load the file 'example2.txt' and print the results. 'example2.txt':

```
@
@ Some objects in the GOODS field
@
unknown $ 189.2207323 $ 62.2357983 $ 26.87 $ 0.32
 galaxy $      *      $ 62.2376331 $ 24.97 $ 0.15
   star $ 189.1409453 $ 62.1696844 $ 25.30 $  *
 *      $ 188.9014716 $     *      $ 25.95 $ 0.20
```

   Load and print:

```
>>> example2 = asciidata.open('example2.txt', null='*', \
                              delimiter='$', comment_char='@')
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy  $            * $  62.2376331 $  24.97 $  0.15
   star  $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
```

4.1.2 create()

This function creates an empty `AsciiData` object in the 'plain' format, which means that the column information is **not** part of the default output. The dimension of the `AsciiData` object as well as the delimiter separating the elements is specified as input.

Usage
create(ncols, nrows, null=None, delimiter=None)

Parameters

Name	Type	Default	Description
ncols	*int*	-	number of columns to be created
nrows	*int*	-	number of rows to be created
null	*string*	'Null'	the character/string representing a null-entry
delimiter	*string*	" "	the delimiter separating the columns

Return
- an `AsciiData` object in the 'plain' format

Examples

1. Create an `AsciiData` object with 3 columns and 2 rows, print the result:

```
>>> example3 = asciidata.create(3,2)
>>> print (example3)
      Null       Null       Null
      Null       Null       Null
```

2. As in 1., but use a different delimiter and NULL value, print the result:

```
>>> example4 = asciidata.create(3,2,delimiter='|', null='<*>')
>>> print (example4)
       <*> |        <*> |        <*>
       <*> |        <*> |        <*>
```

4.1.3 createSEx()

Usage
createSEx(ncols, nrows, null=None, delimiter=None)

Parameters

Name	Type	Default	Description
ncols	*int*	-	number of columns to be created
nrows	*int*	-	number of rows to be created
null	*string*	'Null'	the character/string representing a null-entry
delimiter	*string*	" "	the delimiter separating the columns

Return
- an `AsciiData` object in the SExtractor catalogue format

Examples

1. Create an `AsciiData` object with 3 columns and 2 rows, print the result:

```
>>> example5 = asciidata.createSEx(3,2)
>>> print example5
# 1  column1
# 2  column2
# 3  column3
      Null       Null       Null
      Null       Null       Null
```

2. As in 1., but use a different delimiter and NULL value, print the result:

```
>>> example6 = asciidata.createSEx(3,2,delimiter='|', null='<*>')
>>> print example6
# 1  column1
# 2  column2
# 3  column3
       <*>|       <*>|       <*>
       <*>|       <*>|       <*>
```

4.2 The AsciiData class

The `AsciiData` class is the central class in the `AstroAsciiData` module. After creating `AsciiData` objects with one of the functions introduced in Sect. 4.1, the returned objects are modified using its methods.

4.2.1 AsciiData data

`AsciiData` objects contain some information which is important to the user and can be used in the processing. Although it is possible, this class data should **never** be changed directly by the user. All book-keeping is done internally such that e.g. the value of `ncols` is adjusted when deleting a column.

Data

filename	*string*	file name associated to the object
ncols	*int*	number of columns
nrows	*int*	number of rows

Examples

1. Go over all table entries an store values:

```
>>> example3 = asciidata.create(100,100)
>>> for cindex in range(example3.ncols):
...     for rindex in range(example3.nrows):
...             example3[cindex][rindex] = do_something(rindex, rindex)
...
```

2. Derive a new filename and save the table to this filename:

```
>>> print example2.filename
example2.txt
>>> newname = example2.filename + '.old'
>>> print newname
example2.txt.old
>>> example2.writeto(newname)
```

4.2.2 AsciiData method get

This method retrieves list members of an `AsciiData` instance. These list members are the `AsciiColumn` instances (see Sect. 4.3), which are accessed via their column name **or** column number.
The method returns only the *reference* to the column, therefore changing the returned `AsciiColumn` instance means also changing the original `AsciiData` instance (see Example 2)! To get a deep copy of an `AsciiColumn` the method `copy` of the `AsciiColumn` class itself (see Sect. 4.3.6) must be used instead.

Usage
adata_column = adata_object[col_spec]

or
adata_column = operator.getitem(adata_object, col_spec)

Parameters

col_spec	*string/int*	column specification, either by column name or column number

Return
- an `AsciiColumn` instance

Examples

1. Retrieve the second column of the table:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> aad_col = example[1]
>>> print aad_col
Column: column2
 1.0
 9.5
 3.5
>>>
```

2. Retrieve the second column of the table. Demonstrate that only a shallow copy (reference) is returned:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> ad_col = operator.getitem(example, 'column1')
>>> print ad_col
Column: column1
    1
    2
    3
>>> ad_col[1] = 'new!'
>>> print ad_col
Column: column1
   1
new!
```

```
   3
>>> print example
#
# most important sources!!
#
   1  1.0  red  23.08932 -19.34509
new!  9.5 blue  23.59312 -19.94546
   3  3.5 blue  23.19843 -19.23571
>>>
```

4.2.3 AsciiData method set

This methods sets list members, which means columns, of an `AsciiData` instance. The list member to be changed is addressed either via its column name or the column number.
Obviously the replacing object must be an `AsciiColumn` instance which contains an equal number of rows. Otherwise an exception is risen.

Usage
adata_object[col_spec] = adata_column
or
operator.setitem(adata_object, col_spec, adata_column)

Parameters

col_spec	*string/int*	column specification, either by column name or column number
adata_column	*AsciiColumn*	the `AsciiColumn` instance to replace the previous column

Return
-

Examples

1. Replace the third row of the table 'exa_1' with the third row of table 'exa_2'. Please note the interplay between the `get-` and the `set`-method of the `AsciiData` class:

```
>>> exa_1 = asciidata.open('some_objects.cat')
>>> exa_2 = asciidata.open('some_objects_2.cat', delimiter='|', comment_char='@', null='*')
>>> print exa_1
#
# most important objects
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> print exa_2
@
@
@
   10| 0.0|  pink | 130.3757| 69.87343
   25| 5.3| green | 130.5931| 69.89343
   98| 3.5|      *| 130.2984| 69.30948
>>> exa_1[2] = exa_2[2]
>>> print exa_1
#
# most important objects
#
    1  1.0   pink   23.08932 -19.34509
    2  9.5  green   23.59312 -19.94546
    3  3.5    Null  23.19843 -19.23571
>>>
```

4.2.4 AsciiData method writeto()

Write the `AsciiData` object to a file. The file name is given in a parameter. Independent of the catalogue format (plain or SExtractor) two parameters control whether the column information and the header comment are also written to the new file.

By default the header comments are always written to the file, the column info only for the SExtractor format.

Usage
adata_object.writeto(filename, colInfo, headComment)

Parameters

Name	Type	Default	Description
filename	*string*	-	the filename to save the `AsciiData` object to
colInfo	*int*	*None*	write column info (= 1) or not(= 0)
headComment	*int*	*None*	write header comment (= 1) or not(= 0)

Return
-

Examples

1. Write an `AsciiData` object to the file 'newfile.txt':

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.writeto('newfile.txt')
>>>
> more newfile.txt
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
```

4.2.5 AsciiData method writetofits()

The method transforms an `AsciiData` instance to a fits-table and stores the fits-table to the disk. The filename is either specified as a parameter or is derived from the filename of the original ascii-table. In the latter case the file

extension is changed to '.fits'.
The module PyFITS (see http://www.stsci.edu/resources/software_hardware/pyfits) must be installed to run this method. The transformation fails if the AsciiData instance contains any Null elements (due to a limitation of the numpy and numarray objects, which are essential for the method).

Usage
aad_object.writetofits(fits_name=None)

Parameters

fits_name	*string*	the name of the fits-file

Return
- the fits file to which the AsciiData instance was written

Examples

1. Store an AsciiData instance as a fits-file, using the default name:

```
test>ls
some_objects.cat
test>python
Python 2.4.2 (#1, Nov 10 2005, 11:34:38)
[GCC 3.3.3 20040412 (Red Hat Linux 3.3.3-7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import asciidata
>>> exa = asciidata.open('some_objects.cat')
>>> fits_name = exa.writetofits()
>>> fits_name
'some_objects.fits'
>>>
test>ls
some_objects.cat  some_objects.fits
test>
```

2. Store an AsciiData instance to the fits-file 'test.fits':

```
test>ls
some_objects.cat
test>python
Python 2.4.2 (#1, Nov 10 2005, 11:34:38)
[GCC 3.3.3 20040412 (Red Hat Linux 3.3.3-7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import asciidata
>>> exa = asciidata.open('some_objects.cat')
>>> fits_name = exa.writetofits('test.fits')
>>> fits_name
'test.fits'
```

```
>>>
test>ls
some_objects.cat  test.fits
test>
```

4.2.6 AsciiData method writetohtml()

The method writes the data of an `AsciiData` instance formatted as the content of an html-table to the disk. Strings used as attributes can be specified for the tags `<tr>` and `<td>`. The name of the html-file is either given as parameter or is derived from the name of the original ascii-table. In the latter case the file extension is changed to '.html'.
The html-table is neither opened nor closed at the beginning and end of the file, respectively. Also column names and other meta information is NOT used in the html.

Usage
aad_object.writetohtml(html_name=None, tr_attr=None, td_attr=None)

Parameters

html_name	*string*	the name of the html-file
tr_attr	*string*	attribute string for the tr-tag
td_attr	*string*	attribute string for the td-tag

Return
- the name of the html-file

Examples

1. Write an `AsciiData` instance to an html-file:

```
>>> exa = asciidata.open('some_objects.cat')
>>> exa.writetohtml()
'some_objects.html'
>>>
test>more 'some_objects.html'
<tr><td>    1</td><td> 1.0</td><td> red</td><td> 23.08932</td><td>-19.34509</td></tr>
<tr><td>    2</td><td> 9.5</td><td>blue</td><td> 23.59312</td><td>-19.94546</td></tr>
<tr><td>    3</td><td> 3.5</td><td>blue</td><td> 23.19843</td><td>-19.23571</td></tr>
test>
```

2. Write an `AsciiData` instance to the html-file 'mytab.tab', using attributes for the tags:

```
>>> exa = asciidata.open('some_objects.cat')
>>> html_name = exa.writetohtml('mytab.tab',tr_attr='id="my_tr"',td_attr='bgcolor="RED"')
>>> print html_name
mytab.tab
```

```
>>>
test>more mytab.tab
<tr id="my_tr"><td bgcolor="RED">    1</td><td bgcolor="RED"> 1.0</td><td bgcolor="RED">
 red</td><td bgcolo
r="RED"> 23.08932</td><td bgcolor="RED">-19.34509</td></tr>
<tr id="my_tr"><td bgcolor="RED">    2</td><td bgcolor="RED"> 9.5</td><td bgcolor="RED">
blue</td><td bgcolo
r="RED"> 23.59312</td><td bgcolor="RED">-19.94546</td></tr>
<tr id="my_tr"><td bgcolor="RED">    3</td><td bgcolor="RED"> 3.5</td><td bgcolor="RED">
blue</td><td bgcolo
r="RED"> 23.19843</td><td bgcolor="RED">-19.23571</td></tr>
test>
```

4.2.7 AsciiData method writetolatex()

The method writes the data of an `AsciiData` instance, formatted as the content of a LATEXtable, to the disk. The name of the LATEXfile is either given as parameter or is derived from the name of the original ascii-table. In the latter case the file extension is changed '.tex'.

Usage
aad_object.writetolatex(latex_name=None)

Parameters

latex_name	*string*	the name of the latex-file

Return
- the name of the latex-file

Examples

1. Write the content of an `AsciiData` instance to 'latextab.tb':

```
>>> exa = asciidata.open('some_objects.cat')
>>> latex_name = exa.writetolatex('latex.tb')
>>> print latex_name
latex.tb
>>>
test>more latex.tb
    1& 1.0& red& 23.08932&-19.34509\\
    2& 9.5&blue& 23.59312&-19.94546\\
    3& 3.5&blue& 23.19843&-19.23571\\
test>
```

4.2.8 AsciiData method sort()

This method sorts the data in an `AsciiData` instance according to the values in a specified column. Sorting in ascending and descending order is possible.

There are two different sorting algorithms implemented. A fast algorithm which is based on recursion can be used for making a single, 'isolated' sort process (`ordered=0`).

However the fast algorithm can break, e.g. when sorting an already sorted table (e.g. descending) in the opposite direction (ascending). Then the maximum recursion depth of python can be reached, causing a failure. In addition, fast recursive algorithms introduce random swaps of rows, which is counterproductive if the desired result of the sort process can only be reached with consequtive sortings on different columns (see examples 3 and 4 below).

In these cases a slower sorting algorithm must be used which is evoked with the parameter `ordered=1`.

Usage
adata_object.sort(colname, descending=0, ordered=0)

Parameters

colname	*string/integer*	the specification of the sort column
descending	*integer*	sort in ascending (= 0) or descending (= 1) order
ordered	*integer*	use the fast (= 0) algorithm or the slow (= 1) which avoids unnecessary row swaps

Return
-

Examples

1. Sort a table in ascending order of the values in the second column:

```
>>> sort = asciidata.open('sort_objects.cat')
>>> print sort
    1     0     1     1
    2     1     0     3
    3     1     2     4
    4     0     0     2
    5     1     2     1
    6     0     0     3
    7     0     2     4
    8     1     1     2
    9     0     1     5
   10     1     2     6
   11     0     0     6
   12     1     1     5
>>> sort.sort(1)
>>> print sort
    1     0     1     1
    6     0     0     3
    9     0     1     5
   11     0     0     6
```

```
     7      0      2      4
     4      0      0      2
    12      1      1      5
     2      1      0      3
    10      1      2      6
     3      1      2      4
     5      1      2      1
     8      1      1      2
>>>
```

2. Use the result from example 1, and sort the table in descending order of the first column:

```
>>> sort.sort(0, descending=1)
>>> print sort
    12      1      1      5
    11      0      0      6
    10      1      2      6
     9      0      1      5
     8      1      1      2
     7      0      2      4
     6      0      0      3
     5      1      2      1
     4      0      0      2
     3      1      2      4
     2      1      0      3
     1      0      1      1
>>>
```

3. Sort the table first along column 3 and then along column 2. The resulting table is sorted along column 2, but in addition it is ordered along column 3 for equal values in column 2. This works only using the slower, ordered sorting algorithm:

```
>>> sort.sort(2, ordered=1)
>>> sort.sort(1, ordered=1)
>>> print sort
    11      0      0      6
     6      0      0      3
     4      0      0      2
     9      0      1      5
     1      0      1      1
     7      0      2      4
     2      1      0      3
    12      1      1      5
     8      1      1      2
    10      1      2      6
     5      1      2      1
     3      1      2      4
>>>
```

4. As the previous example, but using the faster, un-ordered sorting algorithm. The result differs from the previous example, since the fast algorithm does **not** preserve the sorting in column 3 for equal values in column 2:

```
>>> sort.sort(2, ordered=0)
>>> sort.sort(1, ordered=0)
>>> print sort
    1     0     1     1
    4     0     0     2
    7     0     2     4
   11     0     0     6
    9     0     1     5
    6     0     0     3
   12     1     1     5
    2     1     0     3
    3     1     2     4
   10     1     2     6
    5     1     2     1
    8     1     1     2
>>>
```

4.2.9 AsciiData method len()

This method defines a length for every `AsciiData` instance, which is the number of columns.

Usage
len(aad_object)

Parameters
-

Return
- the length of the `AsciiData` instance

Examples

1. Determine and print the length of an `AsciiData` instance:

```
>>> exa = asciidata.open('some_objects.cat')
>>> print exa
#
# most important objects
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> length = len(exa)
```

```
>>> print length
5
>>>
```

4.2.10 AsciiData iterator type

This defines an iterator over an `AsciiData` instance. The iteration is finished after `aad_object.ncols` calls and returns each column in subsequent calls. Please note that it is **not** possible to change these columns.

Usage
for iter in aad_object:
... < *do something* >

Parameters
-

Return
-

Examples

1. Iterate over an `AsciiData` instance and print each column name:

```
>>> exa = asciidata.open('sort_objects.cat')
>>> for col in exa:
...     print col.colname
...
column1
column2
column3
column4
>>>
```

4.2.11 AsciiData method append()

Invoking this method is the formal way to append an new column to and `AsciiData` object. When created there are only `Null` entries in the new column. The alternative way is just to specify a column with an unknown name (see Sect. 3.1).

Usage
adata_object.append(col_name)

Parameters

col_name	*string*	the name of the new column

Return

- the number of the columns created

Examples

1. Append a new column 'newcolumn' to the AsciiData object:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy  $            * $  62.2376331 $  24.97 $  0.15
   star  $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
>>> cnum = example2.append('newcolumn')
>>> print cnum
5
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32 $          *
 galaxy  $            * $  62.2376331 $  24.97 $  0.15 $          *
   star  $  189.1409453 $  62.1696844 $  25.30 $     * $          *
       * $  188.9014716 $           * $  25.95 $  0.20 $          *
```

4.2.12 AsciiData method str()

This methods converts the whole AsciiData object into a string. Columns are separated with the delimiter, empty elements are represented by the Null-string and the header is indicated by a comment-string at the beginning. In this method the class object appears as a function argument and the method call is different from the usual form such as in Sect. 4.2.11

Usage
str(adata_object)

Parameters
-

Return
- the string representing the AsciiData object

Examples

1. Print an AsciiData object to the screen:

```
>>> print str(example2)
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy  $            * $  62.2376331 $  24.97 $  0.15
   star  $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
```

2. Store the sting representation of an `AsciiData` object:

```
>>> big_string = str(example2)
>>> print big_string
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy  $            * $  62.2376331 $  24.97 $  0.15
   star  $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
```

4.2.13 AsciiData method del

This method deletes a column specified either by its name or by the column number. Also this method call is slightly different from the usual form such as in Sect. 4.2.11 or 4.2.14.

Usage
del adata_obj[col_spec]

Parameters

col_spec	*string/int*	column specification either by name or by the column number

Return
-

Examples

1. Delete the column with name 'column5':

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy  $            * $  62.2376331 $  24.97 $  0.15
```

```
   star  $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
>>> del example2['column5']
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87
 galaxy  $            * $  62.2376331 $  24.97
   star  $  189.1409453 $  62.1696844 $  25.30
       * $  188.9014716 $           * $  25.95
```

2. Delete the second column:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy  $            * $  62.2376331 $  24.97 $  0.15
   star  $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
>>> del example2[1]
>>> print example2
@
@ Some objects in the GOODS field
@
unknown  $  62.2357983 $  26.87 $  0.32
 galaxy  $  62.2376331 $  24.97 $  0.15
   star  $  62.1696844 $  25.30 $     *
       * $           * $  25.95 $  0.20
```

4.2.14 AsciiData method delete()

This method deletes rows in an `AsciiData` object. The rows to be deleted are specified in the parameters as start index and index of the first row **not** to be deleted. Deletes exactly one row if just the start value is given.

Usage
adata_obj.delete(start, end=start+1)

Parameters

start	*int*	the first row to be deleted
end	*int*	the first row **not** to be deleted

Return

-

Examples

1. Delete only row with index 1:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.delete(1)
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
```

2. Delete the row with index 0 and 1:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
       * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.delete(0,2)
>>> print example2
@
@ Some objects in the GOODS field
@
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
```

4.2.15 AsciiData method strip()

The method removes leading or trailing table rows which are either empty or superfluous. Superfluous rows are marked by the argument given to the method.

Usage

adata_obj.strip(x=None)

Parameters

x *int/float/string* filling value which indicates a superfluous entry

Return

-

Examples

1. Remove all empty rows from the table:

```
>>> print example
#
#
#
 Null  Null  Null  Null
    0     0  Null     1
 Null     9     8     0
 Null  Null  Null  Null
 Null  Null  Null  Null
>>> example.strip()
>>> print example
#
#
#
    0     0  Null     1
 Null     9     8     0
>>>
```

2. Remove all rows which contain **only** the value -1 from the table:

```
>>> print example
#
#
#
   -1    -1    -1    -1
    0     0    -1     1
   -1     9     8     0
   -1    -1    -1    -1
   -1    -1    -1    -1
>>> example.strip(-1)
>>> print example
#
#
#
```

```
     0     0    -1     1
    -1     9     8     0
>>>
```

4.2.16 AsciiData method lstrip()

The method removes all table rows which are either empty or superfluous from the top (≡ **l**eft) of the table. Superfluous rows are marked by the argument given to the method.

Usage
adata_obj.lstrip(x=None)

Parameters

x	*int/float/string*	filling value which indicates a superfluous entry

Return
-

Examples

1. Remove all empty rows from the top of the table:

```
>>> print example
#
#
#
 Null  Null  Null  Null
    0     0  Null     1
 Null     9     8     0
 Null  Null  Null  Null
 Null  Null  Null  Null
>>> example.lstrip()
>>> print example
#
#
#
    0     0  Null     1
 Null     9     8     0
 Null  Null  Null  Null
 Null  Null  Null  Null
```

2. Remove all rows which contain **only** the value -1 from the top of the table:

```
>>> print example
```

```
#
#
#
   -1    -1    -1    -1
    0     0    -1     1
   -1     9     8     0
   -1    -1    -1    -1
   -1    -1    -1    -1
>>> example.lstrip(-1)
>>> print example
#
#
#
    0     0    -1     1
   -1     9     8     0
   -1    -1    -1    -1
   -1    -1    -1    -1
>>>
```

4.2.17 AsciiData method rstrip()

The method removes all table rows which are either empty or superfluous from the bottom (≡ **r**ight) of the table. Superfluous rows are marked by the argument given to the method..

Usage
adata_obj.rstrip(x=None)

Parameters

x	*int/float/string*	filling value which indicates a superfluous entry

Return
-

Examples

1. Remove all empty rows from the bottom of the table:

```
>>> print example
#
#
#
 Null  Null  Null  Null
    0     0  Null     1
 Null     9     8     0
 Null  Null  Null  Null
```

```
 Null  Null  Null  Null
>>> example.rstrip()
>>> print example
#
#
#
 Null  Null  Null  Null
    0     0  Null     1
 Null     9     8     0
>>>
```

2. Remove all rows which contain **only** the value −1 from the bottom of the table:

```
>>> print example
#
#
#
   -1    -1    -1    -1
    0     0    -1     1
   -1     9     8     0
   -1    -1    -1    -1
   -1    -1    -1    -1
>>> example.rstrip(-1)
>>> print example
#
#
#
   -1    -1    -1    -1
    0     0    -1     1
   -1     9     8     0
>>>
```

4.2.18 AsciiData method find()

The method determines the column number for a given column name. The value -1 is returned if a column with this name does not exist.

Usage
adata_obj.find(col_name)

Parameters

col_name	*string*	the name of the column

Return

- the column number or -1 if the column does not exist

Examples

1. Search for the column with name 'column3':

   ```
   >>> example2 = asciidata.open('example2.txt', null='*', \
                                 delimiter='$', comment_char='@')
   >>> cnum = example2.find('column2')
   >>> cnum
   1
   >>>
   ```

2. Search for the column with the name 'not_there':

   ```
   >>> example2 = asciidata.open('example2.txt', null='*', \
                                 delimiter='$', comment_char='@')
   >>> cnum = example2.find('not_there')
   >>> cnum
   -1
   >>>
   ```

   Obviously the `AsciiData` object example2 does not have a column with this name.

4.2.19 AsciiData method flush()

The method updates the associated file with the newest version of the `AsciiData` object.

Usage
adata_obj.flush()

Parameters
-

Return
-

Examples

1. Manipulate an `AsciiData` object and update the file:

   ```
   work>more example.txt
   #
   # Some objects in the GOODS field
   ```

```
#
unknown  189.2207323  62.2357983  26.87  0.32
 galaxy  189.1408929  62.2376331  24.97  0.15
   star  189.1409453  62.1696844  25.30  0.12
 galaxy  188.9014716  62.2037839  25.95  0.20
work>python
Python 2.4.2 (#5, Oct 21 2005, 11:12:03)
[GCC 3.3.2] on sunos5
Type "help", "copyright", "credits" or "license" for more information.
>>> import asciidata
>>> example = asciidata.open('example.txt')
>>> del example[4]
>>> example.flush()
>>>
work>more example.txt
#
# Some objects in the GOODS field
#
unknown  189.2207323  62.2357983  26.87
 galaxy  189.1408929  62.2376331  24.97
   star  189.1409453  62.1696844  25.30
 galaxy  188.9014716  62.2037839  25.95
```

4.2.20 AsciiData method info()

The method returns an informative overview on the `AsciiData` object as a string. This overview gives the user a quick insight into e.g. the column names of the object.

The focus of the method clearly is the use in interactive work. All information provided can be retrieved by `AsciiColumn` methods in a machine readable format as well.

The overview contains:

- the name of the file associated to the `AsciiData` object;
- the number of columns;
- the number of rows;
- the delimiter to separate columns;
- the representing Null-values;
- the comment string.

In addition, for every column the column name, type, format and Null-representation is given.

Usage

adata_object.info()

Parameters
-

Return
-

Examples

1. Print the information on an `AsciiData` object onto the screen:

```
>>> example = asciidata.open('example.txt')
>>> print example.info()
File:       example.txt
Ncols:      4
Nrows:      4
Delimiter:  None
Null value: ['Null', 'NULL', 'None', '*']
Comment:    #
Column name:        column1
Column type:        <type 'str'>
Column format:      ['% 7s', '%7s']
Column null value : ['Null']
Column name:        column2
Column type:        <type 'float'>
Column format:      ['% 11.7f', '%12s']
Column null value : ['Null']
Column name:        column3
Column type:        <type 'float'>
Column format:      ['% 10.7f', '%11s']
Column null value : ['Null']
Column name:        column4
Column type:        <type 'float'>
Column format:      ['% 5.2f', '%6s']
Column null value : ['Null']
```

4.2.21 AsciiData method insert()

This method inserts empty rows into all columns of the `AsciiData` object. The first parameters gives the number of rows to be inserted. The second parameter controls where exactly the new, empty rows are positioned. It specifies the index of the first, empty row. By default the rows are inserted at the table start.

Usage
adata_object.insert(nrows, start=0)

Parameters

nrows	*int*	number of rows to be inserted
start	*int*	index position of the first inserted column

Return

-

Examples

1. Insert two rows such that the first row will have the index 1:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.insert(2,1)
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
        * $            * $           * $      * $     *
        * $            * $           * $      * $     *
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
```

4.2.22 AsciiData method newcomment_char()

The method defines a new comment string for an `AsciiData` object.

Usage

adata_object.newcomment_char(comment_char)

Parameters

comment_char	*string*	the string to indicate a comment

Return

-

Examples

1. Change the comment sign from '@' to '!!':

```
>>> print example2
```

```
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.newcomment_char('!!')
>>> print example2
!!
!! Some objects in the GOODS field
!!
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
```

4.2.23 AsciiData method newdelimiter()

This method specifies a new delimiter for an `AsciiData` object.

Usage
adata_object.newdelimiter(delimiter)

Parameters

delimiter	*string*	the new delimiter to separate columns

Return
-

Examples

1. Change the delimiter sign from '$' to '<>':

```
>>> print example2
!!
!! Some objects in the GOODS field
!!
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.newdelimiter('<>')
>>> print example2
!!
!! Some objects in the GOODS field
!!
unknown   <>  189.2207323 <>  62.2357983 <>  26.87 <>  0.32
 galaxy   <>            * <>  62.2376331 <>  24.97 <>  0.15
```

```
    star   <>  189.1409453 <>  62.1696844 <>  25.30 <>     *
         * <>  188.9014716 <>           * <>  25.95 <>  0.20
```

4.2.24 AsciiData method newnull()

The method specifies a new string to represent Null-entries in an `AsciiData` object.

Usage
adata_object.newnull(newnull)

Parameters

newnull	*string*	the representation for Null-entries

Return
-

Examples

1. Change the Null representation from '*' to 'NaN':

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.newnull('NaN')
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $          NaN $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $   NaN
      NaN $  188.9014716 $         NaN $  25.95 $  0.20
```

4.2.25 AsciiData method toplain()

Change the format of the `AsciiData` object to 'plain'. As a consequence the column info (names, units and comments) are no longer part of the output when e.g. writing the object to a file.

Usage
adata_object.toplain()

Parameters

-

Return

-

Examples

1. Load an `AsciiData` object in the SExtractor format, change to plain format and check the output.

```
>>> SExample = asciidata.open('SExample.cat')
>>> print SExample
# 1  NUMBER  Running object number
# 2  XWIN_IMAGE  Windowed position estimate along x  [pixel]
# 3  YWIN_IMAGE  Windowed position estimate along y  [pixel]
# 4  ERRY2WIN_IMAGE  Variance of windowed pos along y [pixel**2]
# 5  AWIN_IMAGE  Windowed profile RMS along major axis  [pixel]
# 6  ERRAWIN_IMAGE  RMS windowed pos error along major axis  [pixel]
# 7  BWIN_IMAGE  Windowed profile RMS along minor axis  [pixel]
# 8  ERRBWIN_IMAGE  RMS windowed pos error along minor axis  [pixel]
# 9  MAG_AUTO  Kron-like elliptical aperture magnitude  [mag]
# 10 MAGERR_AUTO  RMS error for AUTO magnitude  [mag]
# 11 CLASS_STAR  S/G classifier output
    1  100.523  11.911  2.783  0.0693  2.078  0.0688 -5.3246  0.0416  0.00    19
    2  100.660  4.872  7.005  0.1261  3.742  0.0989 -6.4538  0.0214  0.00    27
    3  131.046  10.382  1.965  0.0681  1.714  0.0663 -4.6836  0.0524  0.00    17
    4  338.959  4.966  11.439  0.1704  4.337  0.1450 -7.1747  0.0173  0.00    25
    5  166.280  3.956  1.801  0.0812  1.665  0.0769 -4.0865  0.0621  0.00    25
>>> SExample.toplain()
>>> print SExample
    1  100.523  11.911  2.783  0.0693  2.078  0.0688 -5.3246  0.0416  0.00    19
    2  100.660  4.872  7.005  0.1261  3.742  0.0989 -6.4538  0.0214  0.00    27
    3  131.046  10.382  1.965  0.0681  1.714  0.0663 -4.6836  0.0524  0.00    17
    4  338.959  4.966  11.439  0.1704  4.337  0.1450 -7.1747  0.0173  0.00    25
    5  166.280  3.956  1.801  0.0812  1.665  0.0769 -4.0865  0.0621  0.00    25
```

4.2.26 AsciiData method toSExtractor()

This method changes the format of the `AsciiData` object to 'SExtractor'. This means that for all output to the screen or to a file the column info precedes the table data.

Usage

adata_object.toSExtractor()

Parameters

-

Return

-

Examples

1. Load a plain `AsciiData` object, change to SExtractor format and write it to a new file. Examine the output on the shell.

```
>>> example = asciidata.open('foo.txt')
>>> print example
    1    stars  1.0
    2 galaxies  2.0
    3     qsos  3.0
>>> example[0].rename('NUM')
>>> example[1].rename('CLASS')
>>> example[2].rename('MAG')
>>> example.toSExtractor()
>>> example.writeto('bar.txt')
>>>
~> more bar.txt
# 1  NUM
# 2  CLASS
# 3  MAG
    1    stars  1.0
    2 galaxies  2.0
    3     qsos  3.0
```

4.2.27 AsciiData method tofits()

The method transforms an `AsciiData` instance to a fits-table extension. This extension might be used with other extensions to build a multi-extension fits-file.
Please use the `AsciiData` method `writetofits()` (see Sect. 4.2.5) to make both, the conversions and storing as a fits-file onto hard disk in one step.
The module `PyFITS` (see http://www.stsci.edu/resources/software_hardware/pyfits) must be installed to run this method. The transformation fails if the `AsciiData` instance contains any `Null` elements (due to a limitation of the `numpy` and `numarray` objects, which are essential for the method).

Usage
aad_object.tofits()

Parameters
-

Return

- a table fits extension

Examples

1. Convert an `AsciiData` object to a fits-table extension and append it to an already existing fits-table (the example is executed in PyRAF):

```
--> catfits exa_table.fits
EXT#  FITSNAME      FILENAME              EXTVE DIMENS       BITPI OBJECT
0     exa_table.fit                                          16
1       BINTABLE    BEAM_1A                     14Fx55R            1
--> exa = asciidata.open('some_objects.cat')
--> tab_hdu = exa.tofits()
--> tab_all = pyfits.open('exa_table.fits', 'update')
--> tab_all.append(tab_hdu)
--> tab_all.close()
--> catfits exa_table.fits
EXT#  FITSNAME      FILENAME              EXTVE DIMENS       BITPI OBJECT
0     exa_table.fit                                          16
1       BINTABLE    BEAM_1A                     14Fx55R            1
2       BINTABLE                                5Fx3R
-->
```

4.3 The AsciiColumn class

The `AsciiColumn` class is the the second important class in the `AstroAsciiData` module. The `AsciiColumn` manages all column related issues, which means that even the actual data is stored in `AsciiColumn` objects. These `AsciiColumn` object are accessed via the `AsciiData` object, either specifying the column name (such as e.g. `adata_object['diff1']`) or the column index (such as e.g. `adata_object[3]`).

4.3.1 AsciiColumn data

`AsciiColumn` objects contain some information which is important to the user and can be used in the processing. Although it is possible, this class data should **never** be changed directly by the user. All book-keeping is done internally.

Data

colname	*string*	file name associated to the object

4.3.2 AsciiColumn method get

This method retrieves one list element of an `AsciiColumn` instance. The element is specified with the row number.

Usage

elem = acol_object[row]
or
elem = operator.getitem(acol_object, row)

Parameters

row	*int*	the row number of the entry to be replaced

Return

- the requested column element

Examples

1. Retrieve and print the first element of the `AsciiColumn` instance which is the third column of the `AsciiData` instance `'exa'`:

```
>>> exa = asciidata.open('some_objects.cat')
>>> print exa
#
# most important objects
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> elem = exa[2][0]
```

```
>>> print elem
 red
>>>
```

4.3.3 AsciiData method set

This methods sets list members, which means elements, of an `AsciiColumn` instance. The list member to be changed is addressed via their row number.

Usage
acol_object[row] = an_entry
or
operator.setitem(acol_object, row, adata_column)

Parameters

row	*int*	the row number of the entry to be replaced
an_entry	*string/integer/float*	the data to replace the previous entry

Return
-

Examples

1. Replace the third entry of the column which is the third column in the `AsciiData` instance `'exa'`:

```
>>> print exa
#
# most important objects
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> exa[2][2] = 'green'
>>> print exa
#
# most important objects
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 green  23.19843 -19.23571
>>>
```

4.3.4 AsciiColumn method len()

This method defines a length of an `AsciiColumn` instance, which equals the number of row in the `AsciiColumn` .

Usage

len(ac_object)

Parameters
-

Return
- the length (= number of rows) of the `AsciiColumn`

Examples

1. Print the length of the fifth column onto the screen:

```
>>> exa = asciidata.open('some_objects.cat')
>>> print exa
#
# most important objects
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> print len(exa[4])
3
>>>
```

4.3.5 AsciiColumn iterator type

This defines an iterator over an `AsciiColumn` instance. The iteration is finished after `acolumn_object.nrows` calls and returns each element in subsequent calls. Please note that it is **not** possible to change these elements.

Usage
for element in acolumn_object:
... $<$ *do something* $>$

Parameters
-

Return
-

Examples

1. Iterate over an `AsciiColumn` instance and print the elements:

```
>>> print exa
#
# most important objects
```

```
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> acol = exa[4]
>>> for element in acol:
...     print element
...
-19.34509
-19.94546
-19.23571
>>>
```

4.3.6 AsciiColumn method copy()

This method generates a so-called *deep copy* of a column. This means the copy is not only a reference to an existing column, but a real copy with all data.

Usage
adata_object[colname].copy()

Parameters
-

Return
- the copy of the column

Examples

1. Copy the column 5 of `AsciiData` object 'example2' to column 2 of `AsciiData` object 'example1'

```
>>> print example1
#
# Some objects in the GOODS field
#
unknown  189.2207323  62.2357983  26.87  0.32
 galaxy            *  62.2376331  24.97  0.15
   star  189.1409453  62.1696844  25.30  0.12
 galaxy  188.9014716  62.2037839  25.95  0.20
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example1[1] = example2[4].copy()
```

```
>>> print example1
#
# Some objects in the GOODS field
#
unknown  0.32  62.2357983  26.87  0.32
 galaxy  0.15  62.2376331  24.97  0.15
   star     *  62.1696844  25.30  0.12
 galaxy  0.20  62.2037839  25.95  0.20
```

4.3.7 AsciiColumn method get_format()

The method returns the format of the `AsciiColumn` object The format description in `AstroAsciiData` is taken from Python. The Python Library Reference (Chapt. 2.3.6.2 in Python 2.5) gives a list of all possible formats.

Usage
adata_object[colname].get_format()

Parameters
-

Return
- the format of the `AsciiColumn` object

Examples

1. Get the format of `AsciiColumn` 0:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2[0].get_format()
'% 9s'
```

4.3.8 AsciiColumn method get_type()

The method returns the type of an `AsciiColumn` object

Usage
adata_object[colname].get_type()

Parameters

-

Return
- the type of the `AsciiColumn`

Examples

1. Get the type of `AsciiColumn` 0:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2[0].get_type()
<type 'str'>
```

4.3.9 AsciiColumn method get_nrows()

This method offers a way to derive the number of rows in a `AsciiColumn` instance.

Usage
acolumn_object.get_nrows()

Parameters
-

Return
- the number of rows

Examples

1. get the number of rows in the column named 'column1':

```
>>> exa = asciidata.open('sort_objects.cat')
>>> exa['column1'].get_nrows()
12
>>> print exa
    1     0     1     1
    2     1     0     3
    3     1     2     4
    4     0     0     2
    5     1     2     1
    6     0     0     3
```

```
    7     0     2     4
    8     1     1     2
    9     0     1     5
   10     1     2     6
   11     0     0     6
   12     1     1     5
>>>
```

4.3.10 AsciiColumn method get_unit()

The method returns the unit of an `AsciiColumn` instance. If there is not unit defined, a string with zero length is returned (‘’)

Usage
acolumn_object.get_unit()

Parameters
-

Return
- the unit of the column

Examples

1. Print the overview of the `AsciiColumn` with index 1:

```
test>more some_objects.cat
# 1 NUMBER          Running object number
# 2 X_Y
# 3 COLOUR
# 4 RA              Barycenter position along world x axis          [deg]
# 5 DEC             Barycenter position along world y axis          [deg]
#
# most important objects
#
1 1.0  red 23.08932 -19.34509
2 9.5 blue 23.59312 -19.94546
3 3.5 blue 23.19843 -19.23571
test>python
Python 2.4.2 (#1, Nov 10 2005, 11:34:38)
[GCC 3.3.3 20040412 (Red Hat Linux 3.3.3-7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import asciidata
>>> exa = asciidata.open('some_objects.cat')
>>> print exa['RA'].get_unit()
deg
>>>
```

4.3.11 AsciiColumn method info()

The method gives an overview on an `AsciiColumn` object including its type, format and the number of elements.

Its focus is on interactive work. All information can also be retrieved by other methods in a machine readable format.

Usage
adata_object[colname].info()

Parameters
-

Return
- the overview on the `AsciiColumn` object

Examples

1. Print the overview of the `AsciiColumn` with index 1:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> print example2[1].info()
Column name:        column2
Column type:        <type 'float'>
Column format:      ['% 11.7f', '%12s']
Column null value : ['*']
```

4.3.12 AsciiColumn method reformat()

The method gives a new format to an `AsciiColumn` object. Please note that the new format does **not** change the column content, but only the string representation of the content. The format description in `AstroAsciiData` is taken from Python. The Python Library Reference (Chapt. 2.3.6.2 in Python 2.5) gives a list of all possible formats.

Usage
adata_object[colname].reformat('newformat')

Parameters

new_format	*string*	the new format of the `AsciiColumn`

Return

-

Examples

1. Change the format of the `AsciiColumn` with index 1:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2[1].reformat('% 6.2f')
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.22 $  62.2357983 $  26.87 $  0.32
 galaxy   $       * $  62.2376331 $  24.97 $  0.15
   star   $  189.14 $  62.1696844 $  25.30 $     *
        * $  188.90 $           * $  25.95 $  0.20
```

4.3.13 AsciiColumn method rename()

The method changes the name on `AsciiColumn` object.

Usage
adata_object[colname].rename('newname')

Parameters

newname	*string*	the filename to save the `AsciiData` object to

Return
-

Examples

1. Change the column name from 'column1' to 'newname':

```
>>> print example2[3].info()
Column name:        column4
Column type:        <type 'float'>
Column format:      ['% 5.2f', '%6s']
Column null value : ['*']

>>> example2[3].rename('newname')
```

```
>>> print example2[3].info()
Column name:         newname
Column type:         <type 'float'>
Column format:       ['% 5.2f', '%6s']
Column null value : ['*']
```

4.3.14 AsciiColumn method tonumarray()

The method converts the content of an `AsciiData` object into a numarray object. Note that this is only possible if there are no Null-entries in the column, since numarray would not allow these Null-entries.

Usage
adata_object[colname].tonumarray()

Parameters
-

Return
- the `AsciiColumn` content in a numarray object.

Examples

1. Convert the column with the index 3 to a numarray object:

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> numarr =  example2[3].tonumarray()
>>> numarr
array([ 26.87,  24.97,  25.3 ,  25.95])
```

4.3.15 AsciiColumn method tonumpy()

The method converts the content of an `AsciiData` object into a numpy object. Columns without Null-entries are converted to numpy array objects, columns with Null-entries become a numpy masked arrays (see numpy manual for details).

Usage
adata_object[colname].tonumpy()

Parameters

-

Return
- the `AsciiColumn` content in a numpy (-masked) object.

Examples

1. Convert the column with index 3 to a numpy object:

```
>>> print example
@
@ Some objects in the GOODS field
@
unknown   $ 189.2207323$ 62.2357983$ 26.87$ 0.32
 galaxy   $        Null$ 62.2376331$ 24.97$ 0.15
   star   $ 189.1409453$ 62.1696844$ 25.30$ Null
      Null$ 188.9014716$       Null$ 25.95$ 0.20
>>> nump = example[3].tonumpy()
>>> print nump
[ 26.87  24.97  25.3   25.95]
```

2. Convert the column with index 2 to a numpy object. This column contains a Null-entry, thus it is converted to a masked array:

```
>>> print example
@
@ Some objects in the GOODS field
@
unknown   $ 189.2207323$ 62.2357983$ 26.87$ 0.32
 galaxy   $        Null$ 62.2376331$ 24.97$ 0.15
   star   $ 189.1409453$ 62.1696844$ 25.30$ Null
      Null$ 188.9014716$       Null$ 25.95$ 0.20
>>> nump = example[2].tonumpy()
>>> type(nump)
<class 'numpy.core.ma.MaskedArray'>
>>> nump[3]
array(data =
 999999,
      mask =
 True,
      fill_value=999999)
```

4.3.16 AsciiColumn method set_unit()

The method sets the unit for a given column. Already existing units are just replaced.

Usage

adata_object[colname].set_unit(acol_unit)

Parameters
acol_unit *string* the new column unit

Return
-

Examples

1. Set a unit for the column *FLAGS*:

```
>>> print sm
# 1  NUMBER  Running object number
# 2  X_IMAGE  Object position along x  [pixel]
# 3  Y_IMAGE  Object position along y  [pixel]
# 4  FLAGS  Extraction flags
    2  379.148  196.397     3
    3  177.377  199.843     4
    1  367.213  123.803     8
>>> sm['FLAGS'].set_unit('arbitrary')
>>> print sm
# 1  NUMBER  Running object number
# 2  X_IMAGE  Object position along x  [pixel]
# 3  Y_IMAGE  Object position along y  [pixel]
# 4  FLAGS  Extraction flags  [arbitrary]
    2  379.148  196.397     3
    3  177.377  199.843     4
    1  367.213  123.803     8
>>>
```

4.3.17 AsciiColumn method set_colcomment()

The method writes a comment for a column into the `AsciiData` header.

Usage
adata_object[colname].set_colcomment(acol_comment)

Parameters
acol_comment *string* the new column comment

Return
-

Examples

1. Set (in this case change) the column comment for the column *FLAGS*:

```
>>> print sm
# 1  NUMBER  Running object number
# 2  X_IMAGE  Object position along x  [pixel]
# 3  Y_IMAGE  Object position along y  [pixel]
# 4  FLAGS  Extraction flags  [arbitrary]
    2  379.148  196.397     3
    3  177.377  199.843     4
    1  367.213  123.803     8
>>> sm['FLAGS'].set_colcomment('Quality numbers')
>>> print sm
# 1  NUMBER  Running object number
# 2  X_IMAGE  Object position along x  [pixel]
# 3  Y_IMAGE  Object position along y  [pixel]
# 4  FLAGS  Quality numbers  [arbitrary]
    2  379.148  196.397     3
    3  177.377  199.843     4
    1  367.213  123.803     8
>>>
```

4.3.18 AsciiColumn method get_colcomment()

The method reads a column comment from an `AsciiData` column.

Usage
adata_object[colname].get_colcomment()

Parameters
-

Return
- the comment string of the column

Examples

1. Read and print the column comment of the column *X_IMAGE*:

```
>>> cocomm = sm['X_IMAGE'].get_colcomment()
>>> print cocomm
Object position along x
>>>
```

4.4 The Header class

The Header class manages the header of an `AsciiData` object. The header contains a list of comments. Any kind of meta-data such as column names are part of the columns and therefore located in the AsciiColumn (see Sect. 4.3) class. The header object is accessed through various methods to e.g. get or set items.

4.4.1 Header method get

The header class contains a method to get individual items from a header instance via their index.

Usage
header_entry = adata_object.header[index]
or
header_entry = operator.getitem(adata_object.header, index)

Parameters

index	*int*	the index of the item to retrieve

Return
- one entry of the header

Examples

1. Retrieve the second entry of this table header:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> header_entry = example.header[1]
>>> print header_entry
 most important sources!!

>>>
```

2. Access the third entry of this table header:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
```

```
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> example.header[2]
'\n'
>>>
```

4.4.2 Header method set

The header class contains a method to set individual items in a header. The item is specified via its index.

Usage
adata_object.header[index] = new_entry
or
header_entry = operator.setitem(adata_object.header, index, new_entry)

Parameters

index	*int*	the index of the item to be set
new_entry	*string*	the new content of the header item

Return
-

Examples

1. Change the second header item:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> example.header[1] = 'a new header entry?'
>>> print example
#
#a new header entry?
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>>
```

2. Change the third header item:

```
>>> print example
#
# most important sources!!
```

```
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> example.header[2] = '   >>> dont forget leading spaces if desired!'
>>> print example
#
# most important sources!!
#   >>> dont forget leading spaces if desired!
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>>
```

4.4.3 Header method del

The header class contains a method to delete individual items in a header. The item is specified via its index.

Usage
del adata_object.header[index]
or
operator.delitem(adata_object.header, index)

Parameters

index	*int*	the index of the item to be deleted

Return
-

Examples

1. Delete the second header item:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> del example.header[1]
>>> print example
#
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>>
```

4.4.4 Header method str()

This method converts the entire AsciiHeader instance into a string. The print command called with an AsciiHeader instance as first parameter also prints the string created using this method str().

Usage
str(adata_object.header)

Parameters
-

Return
- the string representation of the AsciiHeader instance

Examples

1. Delete the second header item:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> print example.header
#
# most important sources!!
#

>>>
```

4.4.5 Header method len()

The method defines the length of an AsciiHeader instance, which equals the number of the comment entries. Please note that empty lines are are counted as well.

Usage
len(adata_object.header)

Parameters
-

Return
- the length of the AsciiHeader instance

Examples

1. Get the length of an `AsciiHeader` :

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> header_length = len(example.header)
>>> header_length
3
>>>
```

4.4.6 Header iterator type

This defines an iterator over an `AsciiHeader` instance. The iteration is finished after `len(adata_object.header)` calls and returns each header element in subsequent calls. Please not that it is **not** possible to change these elements.

Usage
for element in adata_object.header:
... < *do something* >

Parameters
-

Return
-

Examples

1. Iterate over an `AsciiHeader` instance and print the elements:

```
>>> print example
@
@Some objects in the GOODS field
@ -classification
@ -RA
@ -DEC
@ -MAG
@ -extent
unknown   $ 189.2207323$ 62.2357983$ 26.87$ 0.32
 galaxy   $        Null$ 62.2376331$ 24.97$ 0.15
   star   $ 189.1409453$ 62.1696844$ 25.30$ Null
      Null$ 188.9014716$       Null$ 25.95$ 0.20
>>> for h_entry in example.header:
```

```
...     print h_entry.strip()
...

Some objects in the GOODS field
-classification
-RA
-DEC
-MAG
-extent
>>>
```

4.4.7 Header method reset()

The method deletes all entries from an `AsciiHeader` instance and provides a clean, empty header.

Usage
adata_object.header.reset()

Parameters
-

Return
-

Examples

1. Reset an `AsciiHeader` instance:

```
>>> print example
#
# most important sources!!
#
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>> example.header.reset()
>>> print example
    1  1.0  red  23.08932 -19.34509
    2  9.5 blue  23.59312 -19.94546
    3  3.5 blue  23.19843 -19.23571
>>>
```

4.4.8 Header method append()

The method appends a string or a list of strings to the header of an `AsciiData` object.

Usage
adata_object.header.append(hlist)

Parameters

hlist	*string*	the list of strings to be appended to the header

Return
-

Examples

1. Change the column name from 'column1' to 'newname':

```
>>> print example2
@
@ Some objects in the GOODS field
@
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.header.append('Now a header line is appended!')
>>> print example2
@
@ Some objects in the GOODS field
@
@ Now a header line is appended!
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
   star   $  189.1409453 $  62.1696844 $  25.30 $     *
        * $  188.9014716 $           * $  25.95 $  0.20
>>> example2.header.append("""And now we try to put
... even a set of lines
... into the header!!""")
>>> print example2
@
@ Some objects in the GOODS field
@
@ Now a header line is appended!
@ And now we try to put
@ even a set of lines
@ into the header!!
unknown   $  189.2207323 $  62.2357983 $  26.87 $  0.32
 galaxy   $            * $  62.2376331 $  24.97 $  0.15
```

```
star   $  189.1409453 $  62.1696844 $  25.30 $     *
     * $  188.9014716 $           * $  25.95 $  0.20
```

Index