AstroAsciiData

User Manual version 0.01

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

30 November 2005

Contents

1 Introduction

ASCII tables are one of the major data exchange formats used in science. In astronomy ASCII tables are used for a variety of things like object lists, line lists or even spectra. Every person working with 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.

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

- read in ASCII tables;
- manipulate table elements;
- save the modified ASCII table;
- combine several tables;
- delete/add rows and columns.

Python (www.python.org) is in the process of becomming the programming language of choice for astronomers, 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.

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 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 a particularly fast module. The focus was rather to maximizing convenience and ensuring a shallow learning curve for the users.

`AstroAsciiData` is a young and fresh software project. 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 2.2 or later and the numarray (http://www.stsci.edu/resources/software_hardware/numarray) module. It was developed on linux (SUSE, redhat) and Solaris 5.8, however there should be no problems installing it on any machine hosting Python.

The current version 0.01 of `AstroAsciiData` is distributed as the source archive `asciidata-0.01.tar.gz` from the `AstroAsciiData` webpage at http://www.stecf.org/software/astroasciidata/. Installing the module is not difficult. Unpack the tarball with:

```
> gunzip asciidata-0.01.tar.gz
> tar -xvf asciidata-0.01.tar
```

Then enter the the unpacked directory and do the usual:

```
> cd asciidata-0.01
> python setup.py install
```

After installation, some Unit Test are executed with:

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

If there are no errors reported in the Unit Tests, the proper working of the module is assured.

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 whoe really want to find out what is behind the module and how it is built. In case that you just want to use the `AstroAsciiData` module, there is no need to look at its inline documentation.

3 All in one chapter

Reading documentation is no fun. Moreover the `AstroAsciiData` module promised to be convenient for users (see Sect. 1). 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!

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=None)

Parameters

filename	*string*	the name of the ASCII table file to be loaded
null	*string*	the character/string representing a null-entry
delimiter	*string*	the delimiter separating the columns
comment	*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='@')
>>> 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()

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

Parameters

ncols	*int*	number of columns to be created
nrows	*int*	number of rows to be created
null	*string*	the character/string representing a null-entry
delimiter	*string*	the delimiter separating the columns

Return
- an `AsciiData` object

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)
       <*> |        <*> |        <*>
       <*> |        <*> |        <*>
```

4.2 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 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 $          *
```

4.2.3 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. 4.2.2

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
```

4.2.4 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. 4.2.2 or 4.2.5.

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 'column1':

```
>>> 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
```

4.2.5 AsciiData method delete()

This method deletes rows in an `AsciiData` object. The rows to be deleted are specified in the parameters.

Usage
adata_obj.delete(start, end)

Parameters

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

Return
-

Examples

1. Delete the 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(1,2)
>>> 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
```

4.2.6 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='@')
>>> cnum = example2.find('column2')
>>> cnum
1
>>>
```

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

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

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

4.2.7 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
```

4.2.8 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. A further use of the information within programmes is **not** recommended, since all information can also be retrieved by other in a machine usable format using other methods. 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']
```

4.2.9 AsciiData method insert()

This method inserts rows into all columns of the `AsciiData` object. The second parameter controls where exactly the new, empty rows are positioned. The

number specified there the first empty row will be .

Usage
adata_object.insert(nrows, start)

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
```

4.2.10 AsciiData method newcomment()

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

Usage
adata_object.newcomment(comment)

Parameters

comment	*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('!!')
>>> 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
```

4.2.11 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
```

4.2.12 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
```

4.2.13 AsciiData method writeto()

Write the `AsciiData` object to a file. The file name is given in a parameter.

Usage
adata_object.writeto(filename)

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

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
```

4.3 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 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
```

4.3.2 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 2.4) 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'
```

4.3.3 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'>
```

4.3.4 AsciiColumn method info()

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

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 : ['*']
```

4.3.5 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 2.4) 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
```

4.3.6 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 : ['*']
```

4.3.7 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. 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
>>> numarr =  example2[3].tonumarray()
>>> numarr
array([ 26.87,  24.97,  25.3 ,  25.95])
```

4.4 The Header class

The Header class manages the header of an `AsciiData` object. As of now, this header contains only a list of comments. Any kind of meta-data such as column names are **not** used. The header object is accessed through the associated `AsciiData` object, such as e.g. `adata_object.header`.

4.4.1 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