URL: http://cxc.harvard.edu/ciao3.4/readbintab.html
Last modified: December 2006

*AHELP for CIAO 3.4*

# readbintab

Context: varmm



## Synopsis

S−Lang function to read a FITS binary table.

## Syntax

```
Struct_Type readbintab( filename )
Struct_Type readbintab( filename, cols )
Struct_Type readbintab( filename, cols, nskip )

Error Return Value: NULL

Arguments:
filename is a String_Type variable
cols is either a String_Type variable or an Int_Type array
nskip is an Int_Type variable
```

## Description

The readbintab() function provides a high−level interface to reading in a FITS binary table. It can be called either directly or indirectly (i.e. when using the readfile() function). The ahelp page for readfile describes the features of this routine that are common to all the "read" functions provided by the Varmm module. This page describes those features that are unique to the readbintab() command.

The filename argument should be a string that contains the name of the file to be read in. It can include Data Model filters (e.g. "ahelp dmsyntax"), so it can be used to filter tables or select a subset of the columns "on the fly". The optional parameter "cols" is a list of column names or numbers to be read, and "nskip" is the number of rows to be skipped. These two parameters are ignored if a DM filter is supplied as part of the first argument. As examples:

```
  chips> e1 = readbintab( "evt2.fits" )
  chips> e2 = readbintab( "evt2.fits[cols time,sky,pi]" )
  chips> e3 = readbintab( "evt2.fits[sky=region(src.reg)][cols time,pi]" )
  chips> e4 = readbintab( "evt2.fits", "1,2", 10 )
```

where the last line reads in only the first two columns of the file and skips the first ten lines.

## Reading in PHA, ARF, and RMF files

The readpha(), readarf(), and readrmf() functions will also read in FITS binary tables, but only if they are PHA, ARF, and RMF files respectively. As discussed in their respective ahelp pages, these functions provide easy access to relevant parts of the file. However, if you want to access the data as it is stored on disk, use the readbintab() function to read them in.

## What does the function return?

The function returns a structure whose fields contain the data read in from the file. If an error occurred − such as the file not being found, or it not containing a table − then NULL is returned instead. The returned structure follows the format of the other "read" functions: metadata − i.e. information about the file − is stored in fields beginning with an underscore character followed by fields containing the image data and coordinate−transformation information. The initial fields are discussed in "ahelp readfile"; here we concentrate on those fields specific to tables.

### Fields specific to tables:

| Field name: | Description: |
|---|---|
| _ncols | Gives the number of columns read from the table. |
| _nrows | Gives the number of rows read from the table. |
| <column1> | Each column in the table is available by its name. The case of the field name matches that stored in the FITS file unless the varmm.caseinsen variable − a field in the Varmm state obect − is set to 1, in which case the field names are converted to lower case. |

For example, if evt2.fits is a level−two ACIS event file, then we would see something like:

```
chips> evt = readbintab( "evt.fits[cols time,ccd_id,sky,pi]" )
chips> print( evt )
_filename        =  evt2.fits
_path            =  /data/analysis/
_filter          =  [cols time,ccd_id,sky,pi]
_filetype        =  4
_header          =  String_Type[671]
_ncols           =  5
_nrows           =  20
time             =  Double_Type[20]
ccd_id           =  Short_Type[20]
x                =  Float_Type[20]
y                =  Float_Type[20]
pi               =  Integer_Type[20]
```

where the Data Model filter ("[cols time,ccd_id,sky,pi]") has been used just to read in the specified columns. In this particular case the event file only contained 20 events (i.e. rows). Note that vector colums such as the "SKY" column are read in as separate columns − here "x" and "y".

# Example 1

### Reading in a FITS table

```
chips> e = readbintab( "evt2.fits" )
chips> print( e )
_filename        =  evt2.fits
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  4
_header          =  String_Type[696]
_ncols           =  18
_nrows           =  39092
time             =  Double_Type[39092]
ccd_id           =  Short_Type[39092]
node_id          =  Short_Type[39092]
expno            =  Integer_Type[39092]
chipx            =  Short_Type[39092]
chipy            =  Short_Type[39092]
tdetx            =  Short_Type[39092]
tdety            =  Short_Type[39092]
detx             =  Float_Type[39092]
dety             =  Float_Type[39092]
x                =  Float_Type[39092]
y                =  Float_Type[39092]
pha              =  Integer_Type[39092]
energy           =  Float_Type[39092]
pi               =  Integer_Type[39092]
fltgrade         =  Short_Type[39092]
grade            =  Short_Type[39092]
status           =  UChar_Type[39092,4]
```

Note that the function reads in vector columns, such as SKY(x,y) and CHIP(chipx,chipy) as individual columns. The status column has been read in as 4 UChar_Type variables per row, since it contains 32 bits. The order of the array is that the first element of the row − i.e. status[*,0] is the most−significant set of 8 bits while status[*,3] contains the least−significant bits of the status column.

If we had used readfile(), we would have gotten the same structure as shown here.

## Example 2

### Reading in a FITS table using a DM filter

```
chips> e = readbintab( "evt2.fits[#row=11:][cols time,sky,energy]" )
chips> print( e )
_filename        =  evt2.fits
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  4
_header          =  String_Type[696]
_ncols           =  4
_nrows           =  39082
time             =  Double_Type[39082]
x                =  Float_Type[39082]
y                =  Float_Type[39082]
energy           =  Float_Type[39082
```

# Example 3

## Reading in a FITS table using column numbers

```
chips> e = readbintab( "evt2.fits", [1,8,10], 10 )
chips> print( e )
_filename        =  evt2.fits
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  4
_header          =  String_Type[696]
_ncols           =  4
_nrows           =  39082
time             =  Double_Type[39082]
x                =  Float_Type[39082]
y                =  Float_Type[39082]
energy           =  Float_Type[39082]
```

Here we used the optional "cols" and "nskip" to mimic the DM filter used in the previous example. Note that the columns could have been specified as

```
"1,8,10"
```

instead of as an array, and that the column numbers use the DM numbering scheme; i.e. as reported by

```
dmlist evt2.fits cols
```

In general, the DM syntax − as discussed in "ahelp dmsyntax" − is preferred since it is more powerful and flexible than using the "cols" and "nskip" parameters.

# Example 4

Here we use readbintab() to read in just the COUNTS column of a PHA file.

```
chips> pha = readbintab("spec.pha[cols counts]")
chips> print(pha)
_filename        =  spec.pha
_path            =  /data/analysis/
_filter          =  [cols channel,counts]
_filetype        =  4
_header          =  String_Type[352]
_ncols           =  1
_nrows           =  1024
COUNTS           =  Integer_Type[1024]
```

If we had used the readfile() command in this case, it would have called readpha() instead of readbintab(). This would have caused an error, since the DM filter removes columns that readpha() requires, e.g. the CHANNEL column.

# Example 5

Here we write a S−Lang script that can be run by slsh. It reads in the time, sky, and energy columns from evt2.fits and then does some simple arithmetic (calculating the average energy).

```
require("varmm");
varmm.caseinsen = 0;
variable dat = readbintab( "evt2.fits[cols time,sky,energy]" );
if ( dat == NULL )
  error( "Error: Unable to read from evt2.fits" );

vmessage( "The event file contains %d rows", dat._nrows );
vmessage( "The average energy is %f", sum(dat.energy) / dat._nrows );
```

## CHANGES IN CIAO 3.1

### Speed enhancements

The time taken to read in a tables which does not contain any array columns (i.e. more than one element per row of the column) has been reduced.

### Reading a file in a directory containing the string '::'

The routines no longer crash when reading a file within a directory whose name contains the string "::".

### Enhanced documentation

The readbintab function is now documented separately from readfile.

## CHANGES IN CIAO 3.0.2

### Stack Underflow errors

It is now possible to use readfile() − or any of the other read functions described here − in an if statement. Prior to CIAO 3.0.2 you could not write something like

```
if ( NULL == readbintab("evt2.fits") ) error("Failed to read file.");
```

since it would result in a "Stack Underflow" error message. This means that many routines that use readfile() − such as Sherpa's load_dataset() and related functions − can also now be used in an if statement such as:

```
if ( 1 != load_dataset(fname) )
  verror( "Unable to load %s.", fname );
```

## CHANGES IN CIAO 3.0

### New field "_filetype"

A new field called "_filetype" has been added to the data structure which describes the type of the file read in. The contents of the field are described in the "Format of data structure" section in "ahelp readbintab".

# Bugs

### Unable to read in data from virtual vector columns

The readbintab() function does not read in virtual vector columns correctly, although it will read in one of the

components of such a column. So

```
variable f1 = readbintab( "evt2.fits[cols eqpos]" );
variable f2 = readbintab( "evt2.fits[cols ra,dec]" );
```

will produce ra and dec columns − e.g. f1.ra, f1.dec, f2.ra, and f2.dec − that are incorrect, whereas

```
variable f3 = readbintab( "evt2.fits[cols ra]" );
variable f4 = readbintab( "evt2.fits[cols dec]" );
```

produces columns f3.ra and f4.dec that are correct. This only affects columns such as EQPOS that are not stored directly in a file but are created "on the fly" by applying a transform to a column that does exist (for Chandra event files, the EQPOS file is created from the SKY column). If the EQPOS column is a "real" column − e.g. as in the output of the wavdetect tool − then readbintab() works correctly.

See the bugs page for the Varmm library on the CIAO website for an up−to−date listing of known bugs.

# See Also

*modules*

varmm

*varmm*

fits_bitpix, readarf, readascii, readfile, readimage, readpha, readrdb, readrmf, writeascii, writefits

---



URL: http://cxc.harvard.edu/ciao3.4/readbintab.html
Last modified: December 2006