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

AHELP for CIAO 3.4

readpha

Context: varmm

Jump to: Description Examples CHANGES IN CIAO 3.1 CHANGES IN CIAO 3.0.2 CHANGES IN CIAO 3.0 Bugs See Also

Synopsis

S−Lang function to read a spectrum in PHA format (both type I and II)

Syntax

```
Struct_Type readpha( filename )

Error Return Value: NULL

Argument:
filename is a String_Type variable
```

Description

The readpha() function provides a high−level interface to reading in a spectrum stored using the PHA format − both type I and type II − described in the OGIP standard. 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 readpha() command.

The filename argument should be a string that contains the name of the file to be read in. Although it can include Data Model filters it is best not to use them when reading in a PHA file, since any filter may well remove needed information from the file. As an example of how the function is used:

```
  chips> spec = readpha( "src.pha" )
```

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 is not in PHA format − 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 PHA files.

Fields for PHA−I files:

Field name:	Description:
_exptime	The exposure time of the observation in seconds.
_ncols	Gives the number of columns.
_nrows	Gives the number of rows. If the file contains grouping information, then this will generally be larger than the length of the channels and counts arrays.
channels	The channel number. If the file has been grouped, then these numbers will not be match the PHA or PI channel value. The length of this array is given by the numgroups field.
counts	The number of counts in this bin. If the file has been grouped, then this array represents the grouped data. The length of this array is given by the numgroups field.
grouping	This gives the grouping column of the PHA file. Its length matches the original number of rows in the file (i.e. the _nrows and numchans fields). The values of this array are +1 for the start of a new bin and −1 for a continuing bin.
qualityflags	Only present if the file contains a QUALITY column. This array gives the quality value of each bin. Its length matches the counts array. Values of 0 indicate good data, non−zero values indicate bad data.
phachans	This field is only present for grouped PHA files. It gives the channel number − in either PI or PHA units − corresponding to the start of each bin. Its length matches the counts array.
areascal	This is an array with 1 element which contains the area scaling. In general this value will be 1.0.
backscal	This gives the value of the BACKSCAL keyword. It is stored in an array with 1 element.
errors	This array gives the contents of the STAT_ERR column in the PHA file. This field is only available if the file does not contain the keyword POISSERR set to true.
background	This one−element array gives the name of the background file associated with this file.
arf	This one−element array gives the name of the ARF associated with this file.
response	This one−element array gives the name of the RMF associated with this file.
numgroups	This one−element array gives the number of groups in the file. If no grouping information is in the file, then the value will equal the _nrows field.
numchans	This one−element array gives the number of channels in the file, before any grouping was applied. If no grouping information is present, then this will be set to 0.

For example, here is the structure returned when a PHA−I file − with no grouping data − is read in:

```
  chips> pha = readpha("source.pha")
  chips> print( pha )
  _filename        =  source.pha
  _path            =  /data/analysis/
  _filter          =  NULL
  _filetype        =  7
  _header          =  String_Type[304]
  _exptime         =  50703.3
  _ncols           =  11
  _nrows           =  1024
  channels         =  Float_Type[1024]
  counts           =  Float_Type[1024]
  grouping         =  Integer_Type[1024]
  areascal         =  1
  backscal         =  7.72066e-07
  errors           =  Float_Type[1024]
  background       =  none
  arf              =  /data/analysis/source.arf
  response         =  /data/analysis/source.rmf
  numgroups        =  1024
  numchans         =  0
```

The format for type−II PHA files is similar; the main changes are extra information related to the gratings and the fact that the structure now contains data for multiple spectra.

Fields for PHA−II files:

Field name:	Description:
_exptime	The exposure time of the observation in seconds.
_ncols	Gives the number of columns.
_nrows	This gives the number of spectra in the PHA−II file.
channels	As for the PHA−I case, except that the array contains the data for all the spectra.
counts	As for the PHA−I case, except that the array contains the data for all the spectra.
grouping	This gives the grouping data for the spectra. It is stored as a two−dimensional array [m,n], with the first axis (m) being the spectrum number (so its length matches the _nrows field) and the second axis is for the columns in each spectrum. The values of this array are +1 for the start of a new bin and −1 for a continuing bin.
binhi	This gives the upper wavelength for each bin.
binlo	This gives the lower wavelength for each bin.
backup	This gives the background value from the BACKGROUND_UP column.
backdown	This gives the background value from the BACKGROUND_DOWN column.
areascal	As for the PHA−I case.
backscal	As for the PHA−I case.
backscup	The background scaling factor for the backup column. This is stored as a one−element array.
backscdn	The background scaling factor for the backdown column. This is stored as a one−element array.
errors	As for the PHA−I case, except that the array contains the data for all the spectra. This field is only available if the file does not contain the keyword POISSERR set to true.
background	The names of the background files for each of the spectra. This is an array with a length of _nrows.
arf	The names of the ARFs for each of the spectra. This is an array with a length of _nrows.
response	The names of the RMFs for each of the spectra. This is an array with a length of _nrows.
numgroups	The number of groups in each spectrum. This is an array with length of _nrows.
numchans	The number of groups in each spectrum. This is an array with length of _nrows. If no grouping information is present, then each element will be set to 0.
order	This array gives the value of the TG_M column from the input file.
tgpart	This array gives the value of the TG_PART column from the input file.

For example, the following is a PHA−II file from a HETG observation.

```
chips> pha2 = readpha("acisf00459N002_pha2.fits")
chips> print( pha2 )
_filename        =  acisf00459N002_pha2.fits
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  8
_header          =  String_Type[190]
_exptime         =  38564.6
_ncols           =  19
_nrows           =  12
channels         =  Float_Type[98304]
counts           =  Float_Type[98304]
grouping         =  Integer_Type[12,8192]
binhi            =  Float_Type[98304]
binlo            =  Float_Type[98304]
```

```
  backup           =  Float_Type[98304]
  backdown         =  Float_Type[98304]
  areascal         =  1
  backscal         =  1
  backscup         =  0.222222
  backscdn         =  0.222222
  errors           =  Float_Type[98304]
  background       =  String_Type[12]
  arf              =  String_Type[12]
  response         =  String_Type[12]
  numgroups        =  Integer_Type[12]
  numchans         =  Integer_Type[12]
  order            =  Integer_Type[12]
  tgpart           =  Integer_Type[12]
```

Example 1

Reading in a PHA−I file using readpha()

```
  chips> pha = readpha( "src.pi.grp" )
  chips> print( pha )
  _filename        =  src.pi.grp
  _path            =  /data/analysis/
  _filter          =  NULL
  _filetype        =  7
  _header          =  String_Type[352]
  _exptime         =  50703.3
  _ncols           =  12
  _nrows           =  1024
  channels         =  Float_Type[127]
  counts           =  Float_Type[127]
  grouping         =  Integer_Type[1024]
  qualityflags     =  Integer_Type[127]
  phachans         =  Integer_Type[127]
  areascal         =  1
  backscal         =  0.000664699
  background       =  /data/analysis/bgnd.pi
  arf              =  /data/analysis/src.arf
  response         =  /data/analysis/src.rmf
  numgroups        =  127
  numchans         =  1024
```

Here we read in a PHA−I file that had been grouped − i.e. it was the output of dmgroup − which is why the length of the counts array (127) does not match the number of rows in the file (1024).

Example 2

Reading in a PHA−I file using readfile()

Since readfile() calls readpha() when given a PHA file, the following command produces the same structure as the previous example.

```
  chips> pha = readfile( "src.pi.grp" )
```

Example 3

Reading in a PHA−II file using readarf()

Here we use readpha() to read in a PHA−II file from a LETG observation using the HRC−I.

```
chips> pha2 = readpha( "hrcf01801N003_pha2.fits" )
Warning: could not find SYS_ERR column
chips> print( pha2 )
_filename        =  hrcf01801N003_pha2.fits
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  8
_header          =  String_Type[190]
_exptime         =  13726.5
_ncols           =  19
_nrows           =  2
channels         =  Float_Type[32768]
counts           =  Float_Type[32768]
grouping         =  Integer_Type[2,16384]
binhi            =  Float_Type[32768]
binlo            =  Float_Type[32768]
backup           =  Float_Type[32768]
backdown         =  Float_Type[32768]
areascal         =  1
backscal         =  1
backscup         =  4.5
backscdn         =  4.5
errors           =  Float_Type[32768]
background       =  String_Type[2]
arf              =  String_Type[2]
response         =  String_Type[2]
numgroups        =  Integer_Type[2]
numchans         =  Integer_Type[2]
order            =  Integer_Type[2]
tgpart           =  Integer_Type[2]
```

CHANGES IN CIAO 3.1

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 readpha 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 == readfile("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_image(imgname) )
  verror( "Unable to load %s as an image.", imgname );
```

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 readfile".

Bugs

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, readbintab, readfile, readimage, readrdb, readrmf, writeascii, writefits

---

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