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

AHELP for CIAO 3.4

readfile

Context: varmm

Synopsis

S−Lang functions to read a data file into a S−Lang variable.

Syntax

```
Struct_Type readfile( filename )
Struct_Type readfile( filename, cols )
Struct_Type readfile( 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

A S−Lang function in the varmm module that reads in data from a file and stores it in a S−Lang variable. The type of the input file is automatically determined; readfile() supports all file types recognised by the CXC Data Model (DM). The varmm module also provides a number of functions for writing data out to the screen and files. The most relevant functions are writecols(), writeascii(), and writefits(). The ascii2fits program may also be of interest.

The filename parameter is a string that contains the name of the data file to be read in, and can contain DM syntax to filter, select columns, or otherwise manipulate the input file before it is read (see 'ahelp dmsyntax'). Note that you cannot use DM syntax when reading ASCII files as it is not supported for that filetype. 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. If used to read in an image, then the optional arguments are ignored. As examples:

```
chips> e1 = readfile("evt.fits");
chips> e2 = readfile("evt.fits[cols time,sky]");
chips> e3 = readfile("evt.fits","1,2",10);
chips> e4 = readfile("evt.fits",[1,2],10);
```

where the last two examples read in only the first two columns and skip the first ten lines in the file. If the filename contains a list of columns using the Data Model syntax − for example

```
evt2.fits[cols time,sky]
```

− then this takes precedence over the "cols" and "nskip" parameters.

Reading in a specific file type

The readfile() routine works by determining the file type and then calling one of the following routines:

File type	Function
ASCII file	readascii(filename [,cols [,nskip]])
FITS binary table	readbintab(filename [,cols [,nskip]])
FITS image	readimage(filename)
PHA file (type I and II)	readpha(filename)
ARF	readarf(filename)
RMF	readrmf(filename)

If you know the type of file that is to be read in, you can use the matching function instead of readfile(). The functions readpha(), readrmf(), readarf(), and readimage() return extra information that they read from the headers of the files, examples being the names of the ARF and RMF for a pha file, and the World Coordinate System information for an image.

Note that readfile() will not read in a RDB file currently; you have to use the readrdb() function for this.

What is returned by these functions?

The functions return a structure whose fields contain the data read in from the file. If an error occurred − such as the file not being found − then NULL is returned instead. In the returned structure, metadata − e.g. the number of rows read in, the file name, or a pointer to the header information − begins with a single underscore character. These fields are stored first in the structure. Following the metadata is the actual contents of the table or image. Here we focus on those fields common to all file types; the ahelp pages for the individual functions discuss the fields specific to each function.

The fields common to both tables and images are:

Field name	Description
_filename	The name of the file, excluding the path or any Data Model filter.
_path	The full path to the file.
_filter	The Data Model filter applied to the file. If no filter was used then this field will be set to NULL.
_filetype	The type of file (as described below).
_header	The FITS header stored as an array of strings. If the file was an ASCII file then this field will be set to NULL.

The _filetype field

In CIAO 3.0 the "_filetype" field was added to indicate the type of the file from which the data was read. An integer value is used to indicate the type:

Number	Symbol	File type

0	NONE	not applicable
1	ASCII	ASCII file
2	RDB	RDB file
3	FITSIMAGE	FITS image
4	FITSBIN	FITS binary table
5	IMH	IRAF IMH file
6	QPOE	IRAF QPOE file
7	PHA_I	PHA type I file
8	PHA_II	PHA type II file
9	ARF	ARF
10	RMF	RMF
11	IMAGE	synonym for FITSIMAGE and IMH

The "Symbol" column gives the name of the S−Lang variable that can be used to check the file type; for example:

```
variable in = readfile(infile);
switch ( in._filetype )
{ case ASCII: vmessage( "file=%s is an ASCII file", infile ); }
{ case ARF:   vmessage( "file=%s contains an ARF", infile ); }
...
```

The _header field

The header information of a FITS file is stored as an array of strings in the _header field of the structure: for ASCII files it is set to NULL. Each FITS card (i.e. keyword, value, and comment), is stored as one item in a string array, for example:

```
chips> arf = readbintab("centre.arf")
chips> print(arf._header)
String_Type[170]
chips> printarr(arf._header[0])
XTENSION= 'BINTABLE'           / binary table extension
chips> printarr(arf._header[[8:9]])
EXTNAME = 'SPECRESP'           / name of this binary table extension
HDUNAME = 'SPECRESP'           / ASCDM block name
```

If you do not wish headers to be read, then you can set the readheader element of the varmm state object to 0 to turn off this behaviour:

```
chips> varmm.readheader = 0
```

Setting it to 1 will restore the default value of reading in the headers:

```
chips> varmm.readheader = 1
```

The remaining contents of the structure depend upon the file type. See the ahelp page for the apropriate function − e.g. "ahelp readimage" for images − for details on these fields.

Example 1

Here we read an ASCII file called "phas.dat", which contains two columns, into a S−lang variable called dat:

```
sherpa> dat = readfile("phas.dat");
```

We then use the print() function to view the contents of this variable:

```
sherpa> print(dat);
_filename        =  phas.dat
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  1
_header          =  NULL
_ncols           =  2
_nrows           =  128
col1             =  Float_Type[128]
col2             =  Float_Type[128]
```

Example 2

Here we use readfile() on a PHA−I file. It calls readpha() internally, once it has found out the file type, and so the return value is the same as if we had used readpha("spec.pha").

```
chips> bar = readpha("spec.pha")
chips> print(bar)
_filename        =  spec.pha
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  7
_header          =  NULL
_exptime         =  51234.2
_ncols           =  11
_nrows           =  1024
channels         =  Float_Type[682]
counts           =  Float_Type[682]
grouping         =  Integer_Type[1024]
qualityflags     =  Integer_Type[682]
phachans         =  Integer_Type[682]
areascal         =  1
backscal         =  1
background       =  none
arf              =  none
response         =  none
numgroups        =  682
numchans         =  1024
```

See "ahelp readpha" for more information on this structure.

Example 3

You are not restricted to accessing column data, since readfile() can also read in FITS images. As well as the image data − stored in the "pixels" field − the structure also contains information about the Physical and World Coordinate System transformations of the image.

```
chips> img = readfile("spec.wmap")
chips> print(img)
_filename        =  spec.wmap
_path            =  /data/analysis/
_filter          =  NULL
_filetype        =  11
_header          =  String_Type[460]
```

```
  _transform       =  TAN-P
  _naxes           =  2
  pixels           =  Short_Type[1024,1024]
  min              =  Double_Type[2]
  max              =  Double_Type[2]
  step             =  Double_Type[2]
  crval            =  Double_Type[2]
  crpix            =  Double_Type[2]
  crdelt           =  Double_Type[2]
```

See "ahelp readimage" for more information on these fields.

Example 4

The readfile() function can also be used in S−Lang scripts run by slsh. In this situation, the varmm module must be loaded before any of the read functions are used.

```
require("varmm");
varmm.caseinsen = 0;
variable dat = readfile( "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

Files are no longer cached

Prior to CIAO 3.0, if a table or image was read in then that data would be cached so that any attempt to re−read the file would lead to the original data being returned, even if the actual file on disk had changed. In CIAO 3.0 this caching was removed for tables, and with the CIAO 3.1 release it has also been removed for images. This change has also resulted in reduced memory use.

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

There are now separate ahelp files for the individual functions, rather than having all the information placed into this document.

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 );
```

Reading BYTE images

Prior to CIAO 3.0.2, images of BYTE type were being stored using a Char_Type array. They are now stored in a UChar_Type array.

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 above.

Format of PHA files

The order of the fields for PHA files has been changed and the exposure time is now stored using the field "_exptime" − previously it was stored in the "exptime" field.

RDB support

The readrdb() routine has been added to allow reading in of RDB files. Note that readfile() does not recognise RDB files, so you have to call readrdb() directly.

Bugs

readfile() does not recognise RDB files

In CIAO 3.2, readfile() will not read in a RDB file; you have to use the readrdb() function for this.

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, readimage, readpha, readrdb, readrmf, writeascii, writefits

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