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

AHELP for CIAO 3.4

readimage

Context: varmm

Jump to: Description Examples CONVERTING BETWEEN COORDINATE SYSTEMS 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 FITS image.

Syntax

```
Struct_Type readimage( filename );

Error Return Value: NULL

Argument:
filename is a String_Type variable
```

Description

The readimage() function provides a high−level interface to reading in a FITS image. 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 readimage() 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 read in an event file which is then binned into an image "on the fly". As examples:

```
  chips> img1 = readimage("img.fits");
  chips> img2 = readimage("img.fits[sky=region(source.reg)]");
  chips> img3 = readimage("evt2.fits[bin sky=region(source.reg)]");
```

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 an image − 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 images. The section "CONVERTING BETWEEN COORDINATE SYSTEMS" after the examples discusses how to calculate the physical and world coordinates of a given pixel.

Fields specific to images (see "ahelp dmimages" for more information on the logical, physical, and world coordinate systems):

Field name:	Description:
_transform	A string giving the type of the world coordinate transformation stored in the image. If no information is available, the field will be set to "none".
_naxes	The number of axes of the image.
pixels	The image data as a S−Lang array. The FITS datatype (as given by the BITPIX keyword) will be converted to the appropriate S−Lang data type. If the FITS image has m by n pixels, as given by dmlist, then the S−Lang array will have dimensions [n,m], as discussed in the following section ("The order of the axes").
min	An array of the minimum physical coordinates of the array. They correspond to the center of the first pixel; i.e. position 1 in the FITS standard. The data is stored in the FITS axis order, so that the first element − min[0] − corresponds to the first (which is often the "X") axis.
max	An array of the maximum physical coordinates of the array. They correspond to the center of the last pixel; i.e. position (m,n) in the FITS standard for a two−dimensional image with m columns (pixels along the "X" axis) and n rows (pixels along the "Y" axis).
step	This array gives the width of a pixel in physical units.

The following fields will only be created if the image contains World Coordinate System (WCS) information:

Field name:	Description:
crval	This array gives the world coordinates of the reference position. The array is stored in FITS order, so that the first element − crval[0] − corresponds to the X axis.
crpix	This gives the physical coordinates of the reference pixel for the world coordinate system.
crdelt	This array gives the size of a single physical pixel in units of the world coordinate system (normally degrees). Please note that the size of a logical pixel − i.e. one pixel of the pixels array − is equal to crdelt * step.

The order of the axes

The most important thing to note is that S−Lang arrays are indexed as in C, so the right−most index loops fastest, whilst FITS images are indexed like FORTRAN, where the left−most index loops fastest. Therefore a two−dimensional image which dmlist reports to have 154 by 48 pixels will be read into a 48 by 154 element S−Lang array. The column of pixels corresponding to the Data Model filter

```
#1=54
```

would be given by

```
img.pixels[*,53]
```

in S−Lang (the Data Model follows the FITS convention of counting arrays from 1 whilst S−Lang counts from 0).

Although the order of the axes is swapped between FITS and S−Lang, please note that other elements of the file are NOT swapped. Element 0 of the min, max, step, crval, crpix, and cdelt arrays refer to the first, often the "X", axis (called AXIS1 in FITS). Similarly, the FITS header information − stored in the _header field − will still be stored using the FITS system.

For example, in the following emap.fits is an exposure map created by mkexpmap. We use a spatial filter to read in a 154 by 48 pixel part of the map; the resulting pixel array has dimensions of [48,154].

```
  chips> img=readfile("emap.img[sky=box(4699.25,4243.75,153.5,47.5,0)]")
  chips> print( img )
  _filename        =  emap.img
  _path            =  /data/analysis/
  _filter          =  [sky=box(4699.25,4243.75,153.5,47.5,0)]
  _filetype        =  11
  _header          =  String_Type[158]
  _transform       =  TAN
  _naxes           =  2
  pixels           =  Float_Type[48,154]
  min              =  Double_Type[2]
  max              =  Double_Type[2]
  step             =  Double_Type[2]
  crval            =  Double_Type[2]
  crpix            =  Double_Type[2]
  crdelt           =  Double_Type[2]
```

There is another illustration of this in Example 2, below.

Example 1

Here we read the image "img.fits" into a S−Lang variable called img. The pixels field contains the image data; the sum() function is used to find the total number of counts in the image.

```
  chips> img = readfile("img.fits")
  chips> print(img)
  _filename        =  img.fits
  _path            =  /data/analysis/
  _filter          =  NULL
  _filetype        =  11
  _header          =  String_Type[460]
  _transform       =  TAN
  _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]
  chips> sum( img.pixels )
  8462
```

Note that we could have used the readfile() command instead of readimage() in this example.

Example 2

When an image is read in, the pixels array uses S−Lang indexing so that the right−most array loops fastest. For the example above this corresponds to the X axis of the image. This is in contrast to FITS, which uses the FORTRAN standard that the left−most index loops fastest. This means that a three−dimensional image which dmlist reports as being i by j by k pixels will be read into a S−Lang array with dimensions [k,j,i]. There is also the fact that S−Lang arrays start counting from 0 whereas the Data Model (following FITS) starts arrays at 1.

As an example, consider an image with 5 columns and 2 rows: i.e. it has 5 pixels along the X axis and 2 pixels along the Y axis. We use the _reshape() function to create this array in S−Lang with the values 1 to 10:

```
  chips> a = _reshape( [1:10], [2,5] )
  chips> a
```

```
Integer_Type[2,5]
chips> print( a )
1       2       3       4       5
6       7       8       9       10
chips> print( a[0,*] )
1
2
3
4
5
chips> print( a[*,2] )
3
8
chips> print( a[1,3] )
9
```

This says that the first row of the image contains the values (1,2,3,4,5), the third column contains the values (3,8) and the second pixel along the X axis and fourth along the Y axis contains the value 9.

We can write this array out to a FITS file, using writefits(), and use dmlist to look at its contents.

```
chips> () = writefits( "test.img", a )
```

Since dmlist uses the FITS standard, the array is listed as number of columns by number of rows and axes start at 1 (rather than at 0 as with S−Lang).

```
unix% dmlist test.img blocks

--------------------------------------------------------------------
Dataset: test.img
--------------------------------------------------------------------

     Block Name                          Type         Dimensions
--------------------------------------------------------------------
Block    1: PRIMARY                        Image      Int4(5x2)
unix% dmlist test.img data,array

--------------------------------------------------------------------
Data for Image Block PRIMARY
--------------------------------------------------------------------

ROW    CELL   PRIMARY[5,2]

     1 [     1     1 ]           1
     1 [     2     1 ]           2
     1 [     3     1 ]           3
     1 [     4     1 ]           4
     1 [     5     1 ]           5
     1 [     1     2 ]           6
     1 [     2     2 ]           7
     1 [     3     2 ]           8
     1 [     4     2 ]           9
     1 [     5     2 ]          10
```

We can check that a[1,3] has been stored as pixel (#1=4,#2=2) by:

```
unix% dmlist "test.img[#1=4,#2=2]" data,clean
#  PRIMARY[1,1]
         9
```

Example 3

The fields following the pixels array contain information on the coordinate mapping of the image; i.e. how to go from pixel number (the logical coordinate system) to physical and world coordinate systems. These coordinate systems are described in "ahelp dmimages" and "ahelp coords".

Here we use readimage() to read in a Chandra ACIS event file, converting it to an image by using a Data Model filter (see "ahelp dmsyntax" for more information on the Data Model syntax).

```
chips> fname = "evt2.fits[bin x=3739.5:4763.2:1,y=3163.5:4187.5:1]"
chips> img = readfile( fname )
chips> print( img )
_filename        =  evt2.fits
_path            =  /data/analysis/
_filter          =  [bin x=3739.5:4763.2:1,y=3163.5:4187.5:1]
_filetype        =  11
_header          =  String_Type[696]
_transform       =  TAN
_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]
```

The image has 1024 by 1024 pixels and goes from 3739.5 to 4763.5 and 3163.5 to 4187.5 in physical units (here the sky or (x,y) coordinate system), where the values give the values of the pixel edges. The min, max, and step arrays contain the information needed to convert from pixel to physical coordinate systems. The img.min and img.max fields give the coordinates of the centers of the first and last pixels − here they are the [0,0] and [1023,1023] elements of the S−Lang array − while step gives the size of a pixel in physical coordinates. So, for this file we get:

```
chips> writeascii( stdout, img.min, img.max, img.step )
3740    4763    1
3164    4187    1
```

which says that the first pixel covers the range (3739.5,3163.5) to (3740.5,3164.5) and the last pixel covers (4762.5,4186.5) to (4763.5,4187.5). The corresponding mapping from physical to world coordinate systems is given in the crval, crpix, and crdelt arrays. For this particular observation we have

```
chips> writeascii( stdout, img.crval, img.crpix, img.crdelt*3600 )
259.254 4096.5  -0.492
67.1953 4096.5  0.492
```

which says that physical coordinate (4096.5,4096.5) corresponds to RA and Declination of (259.254,67.1953) degrees with a pixel scale of (−0.492,0.492) arcsec. Note that writeascii() uses the default floating−point format for writing out data (normally it is "%f", which can be changed by the set_float_format() function). To see the RA "crval" value in greater precision you can use something like:

```
chips> vmessage( "%15.11f", img.crval[0] )
259.25419123646
```

Example 4

Here we read in the same region as the previous example but bin by 2 instead of 1, so that one pixel of the image now corresponds to 2x2 physical pixels.

```
chips> fname2 = "evt2.fits[bin x=3739.5:4763.2:2,y=3163.5:4187.5:2]"
chips> img2 = readfile( fname2 )
chips> print( img2 )
_filename        =  evt2.fits
_path            =  /data/analysis/
_filter          =  [bin x=3739.5:4763.2:2,y=3163.5:4187.5:2]
_filetype        =  11
_header          =  String_Type[696]
_transform       =  TAN
```

```
_naxes           =  2
pixels           =  Short_Type[512,512]
min              =  Double_Type[2]
max              =  Double_Type[2]
step             =  Double_Type[2]
crval            =  Double_Type[2]
crpix            =  Double_Type[2]
crdelt           =  Double_Type[2]
```

The image is now only 512 by 512 and the coordinate mapping information now looks like:

```
chips> writeascii( stdout, img2.min, img2.max, img2.step )
3740.5  4762.5  2
3164.5  4186.5  2
chips> writeascii( stdout, img2.crval, img2.crpix, img2.crdelt*3600 )
259.254 4096.5  −0.492
67.1953 4096.5  0.492
```

Note that the only change to the values given in the previous example is the step array, which is now 2 rather than 1. The crdelt array has not changed since this gives the size of one physical pixel rather than the size of one image pixel (which in this example is composed of 2x2 physical pixels).

Example 5

In the following example we add together two images − img1.fits and img2.fits − and then use writefits() to write the combined image out to img3.fits. This is written as a S−Lang script which can be run using slsh. We therefore have to load the Varmm module − to make the readimage() and writefits() functions available − and ensure we use valid S−Lang syntax (i.e. ensure all statements end in a semicolon and variables are declared).

```
require("varmm");
variable i1 = readimage( "img1.fits" );
if ( i1 == NULL )
  error( "Error: Unable to read in img1.fits" );
variable i2 = readimage( "img2.fits" );
if ( i2 == NULL )
  error( "Error: Unable to read in img2.fits" );

variable total = i1.pixels + i2.pixels;
if ( 0 != writefits( "img3.fits", total ) )
  error( "Unable to create img3.fits" );
```

Unlike dmtcalc, this example does not preserve the header information; it could be expanded to include that information, however, since writefits() allows you to write out a FITS header.

CONVERTING BETWEEN COORDINATE SYSTEMS

In the following we consider a two−dimensional image with m columns and n rows. If the image was read into a variable called img then the img.pixels array would have dimensions [n,m]. Coordinates listed using [] are given in the S−Lang system, where the right−most axis loops fastest and so corresponds to the first axis of an image, while those in () are given in the FITS system, where the left−most index represents the first axis.

From the logical to physical coordinate system

The structure returned by readimage() contains min, max, and step fields which tell you how to map from the logical system (i.e. the pixel coordinates) to the associated physical system. For Chandra images the physical system is normally the SKY coordinate system (as described in "ahelp coords").

The physical coordinates (px,py) of pixel img.pixels[j,i] are given by:

```
px = img.min[1] + i * img.step[1]
```

and

```
py = img.min[0] + j * img.step[0]
```

These coordinates refer to the center of the pixel.

Coordinate conversions:

S−Lang coordinate:	px	py
[0,0]	img.min[0]	img.min[1]
[0,m−1]	img.min[0] + (m−1)*img.step[0]	img.min[1]
[n−1,0]	img.min[0]	img.min[1] + (n−1)*img.step[1]
[n−1,m−1]	img.min[0] + (m−1)*img.step[0]	img.min[1] + (n−1)*img.step[1]

Note that for these particular coordinates, which correspond to the four corner pixels, we could have used the contents of the img.max array rather than calculate the values using the img.min and img.step arrays.

From the physical to world coordinate system

Once the pixel coordinate has been transformed to the physical system, it can then be changed into the world coordinate system if the crval, crpix, and crdelt fields exist. For Chandra images this is normally the EQPOS coordinate system, which, as described in "ahelp coords", gives the RA and Declination of the position.

The world coordinates (wx,wy) of pixel img.pixels[j,i] are given by:

```
wx = img.crval[0] + (px − img.crpix[0]) * img.crdelt[0]
```

and

```
wy = img.crval[1] + (py − img.crpix[1]) * img.crdelt[1]
```

where (px,py) are calculated using the formula given in above. Note that in this step we do not have to reverse the order of the arrays, unlike the calculation for (px,py).

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 cacheing was removed for tables, and with the CIAO 3.1 release it has also been removed for images.

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 readimage function is now documented separately from readfile. A discussion of the axis order of image data and the conversion between coordinate systems has been added.

CHANGES IN CIAO 3.0.2

Stack Underflow errors

It is now possible to use readimage() in an if statement. Prior to CIAO 3.0.2 you could not write something like

```
  if ( NULL == readimage("img.fits") ) error("Failed to read file.");
```

since it would result in a "Stack Underflow" error message. This means that many routines that use readimage() − 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 in "ahelp readbintab".

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, readpha, readrdb, readrmf, writeascii, writefits

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