CXC-DM-012

CXC Data Model

Chandra Data Model Manual, Part 3: User Reference

CIAO2.2 Edition
Chandra X-ray Center
October 22, 2001

Contents

1 Introduction

This document describes the CIAO DM ('Data Model') filtering and binning language. It has significant overlap with the on-line ahelp files found by typing 'ahelp dm' in CIAO. Users may wish to first read the User Introduction which gives simple examples, and to use this document in conjuction with the Toolkit User Guide which describes the full DM syntax.

2 Virtual File Syntax Reference

The virtual file syntax is a way of describing a filtered version of a DM block (a single table or image). For instance, if evt.fits is a FITS file with a binary table called EVENTS in it,

```
evt.fits[events][pha=10:200,det=circle(4096,4096,50)][bin det=8]
   [opt null=NaN,type=r4][img8]
```

defines an image made by making a histogram on the "det" vector column and putting the results in a 4-byte real image, binning by 8 (1 output image pixel is a range of 8 in each direction in values of the det coordinate in the input table), with the histogram made counting only rows in the table satisfying the filter, and with pixels in the image outside the filter circle set to NaN.

- The virtual block specification consists of a string consisting of a 'base dataset' name and a series of qualifier sections. In the example, 'evt.fits' is the base dataset.
- Each qualifier section is a string delimited by square parentheses [].
- The possible qualifiers are:
  - block identifier
  - filter-command
  - column select/bin-command
  - name-command
  - option command

  Any qualifier may be omitted, but those which appear must appear in this order.
- There should be at most one of each type of qualifier in a specification.

- A virtual file specifies either a table or an image. Whether a given string is a table spec or an image spec depends on external context, and cannot in general be determined from the string itself.
- A column-select or bin command qualifier may be recognized by the fact that it begins with the text ‘columns’, ‘bin’ or ‘bin/f’ followed by a space followed by further text. In the above example, we force making an image by using the bin directive ‘[bin det=8]’.
- An option qualifier may be recognized by the fact that it begins with the text ‘opt’ followed by a space followed by further text. In the example, ‘[opt null=NaN,type=r4]’.
- A block identifier may be recognized because it consists of only a single word and is the first qualifier. In the example, ‘[events]’.
- A name-command may be recognized because it consists of either a single word, or a single word followed by a space and other text, and it cannot be a block identifier because it is not the first qualifier, and it cannot be a select/bin command because the word is not ‘select’ or ‘bin’. In the example, ‘[img8]’ is the name command.
- A filter command can be recognized because it does not match any of the other qualifiers. In the example, ‘[pha=10:200,det=circle(4096,4096,50)]’ is the filter qualifier. We could also have said more explicitly ‘[filter pha=10:200,det=circle(4096,4096,50)]’.

3 Base dataset

The base dataset identifies the input dataset (file or files).

- The base dataset name is the name of a file on disk. (Note that in the case of the IRAF/QPOE kernel, this file may be a directory). The full filename should be given including any extension. URLs are not currently supported.
- If no base dataset name is given and no dataset is otherwise specified to the software, the meaning is undefined.

4 Block identifier

```
[name]
[#n]
```

The block identifer specifies the input DM block (table or image) within the selected dataset. Often, it's omitted and the DM tries to guess which block in the dataset you are likely to be interested in. When in doubt, though, you can always specify the block by name. To see what the block names are in the dataset, use 'dmlist *filename* blocks'.

- The block identifier, if present, must be the first qualifier section.
- The block identifier consists of a single word which may contain letters, digits, and the characters . (dot), - (minus), + (plus), / (slash), _ (underbar), # (hash), $ (dollar), ; (semicolon). Some other characters may also be accepted, but it may not contain whitespace or the characters comma (,), bar (|), square parens ([,]), equals (=), colon (:).
- If the block identifier consists only of digits, (and represents an integer n which is less than the number of DM tables in the dataset), the interpretation is that it refers to the nth DM table in the dataset, counting from 1. The block identifer [0] is illegal.
- If the block identifier is empty, i.e. [], or not present, the first 'interesting' block in the dataset is implied. For FITS, this is the first block with positive NAXIS. So in a FITS binary table with a null primary header, the following

  ```
  rh1012.fits
  rh1012.fits[]
  rh1012.fits[2]
  ```

  are all identical.
- Otherwise, the block identifier is interpreted as the name of an DM block within the dataset.

5 Table Filtering

5.1 Description

If the underlying block is a table, the filter directive selects rows from the table. If the underlying block is an image, the filter selects an image subsection, discussed in a later section of this document.

```
[filter nameA=min1:max1,min2:max2,...minN,maxN]
```

```
[filter nameA=min1:max1,...minN,maxN,nameB=min1:max1,...]
[nameA=min1:max1,...minN,maxN,nameB=min1:max1,...]
[filter nameA=shape(parameters),nameB=REGION(region_filename)]
[filter @filter.lis]
[filter nameA<max,nameA>min,nameA!=val]
[filter (nameA=min1:max1)||(nameA=min3:max3,nameB=min3:max3)]
[exclude nameA=min1:max1,min2:max2,..,nameB=...]
```

If the words 'filter(space)' and 'exclude(space)' are not present, the parser recognizes the syntax as a filter directive because of the equals sign. (With the equals sign, it can't be a block name; but it can't be a cols or bin directive either because the 'col(space)','bin(space)' are not optional in those cases.)

5.2 Table filtering on columns with real data type

There are a number of ways to filter a DM block (table or image in a file). In this section we describe table filtering.To see which column names you can filter on, use

```
dmlist a.fits cols
```

to display the columns in the main block of file a.fits. The simplest kind of filter is a range filter

```
[filter energy=1000:2000]
```

which specifies that the DM should only see rows in the input file which satisfy 1000.0 <= energy <2000.0. You can use this filter by appending it to a filename or block name in any of the CIAO tools:

```
dmcopy "a.fits[filter energy=1000:2000]" b.fits
```

```
dmcopy "a.fits[events][filter energy=1000:2000]" b.fits
```

Note that the string "filter(space)" is optional:

```
dmcopy "a.fits[energy=1000:2000]" b.fits
```

You can filter on multiple quantities:

```
dmcopy "a.fits[energy=1000:2000,time=5410300:5410320]" b.fits
```

You can also filter on multiple ranges for each quantity:

```
dmcopy "a.fits[energy=1000:2000,4000:8000,grade=0,2:4,6]" b.fits
```

This filter accepts rows which have energies between either 1000 and 2000 or 4000 and 8000, and grades equal to 0, 2 to 4, or 6. Note that you can leave out the colon if the min and max of a range are the same, so 0:0 becomes just 0. If you want to express "less than" and "greater than", you can just retain the colon but omit the min or max:

```
[energy=:4000]
```

means accept energies up to 4000;

```
[detx=:511,513:]
```

means accept all values of detx except 511.0 <= detx < 513.0.

5.3 Table filtering on columns with integer data types

The interpretation is a little different depending on whether the table column has integer or real data type. For an integer data type column,

```
[filter energy=4000:5000]
```

means (4000 <= energy <= 5000), in other words both ends of the range are included.

The syntax also supports the special pseudo-column #row, the row number of the unfiltered file:

```
[filter #row=100:200]
```

5.4 Table filtering on columns with character string data types

Filtering is a little more restricted with character string columns. You can only use the colon syntax, for example:

```
[filter shape=m:n,point,rectangle,s:z]
```

Be careful: the range m:n includes everything beginning with m, and it includes the letter n, but not other strings beginning with n: for example, "ngc" is not within m:n since in an ASCII ordering ngc >n. The comparison is case-sensitive.

5.5 Table filtering on columns with bit data type

Columns with bit data type are a special case. The most common example is the STATUS column in the event files, which is 32 bits wide. Suppose for simplicity you instead have a status column with only 6 bits, and wish to accept rows with the 3rd bit from the end set and the end bit equal to zero. A simple numeric filter of the type described above would have to be very complicated to describe all the numeric values for which these bits have the desired values; instead, the user supplies a 'bitmask string':

```
[filter status=xxx1x0]
```

The string may only contain the characters 1 (corresponding bit must be set), 0 (bit must not be set) and x (wild card: bit may have any value).

The bit pattern display convention here is that the rightmost bit displayed (with the value 0 in the example above) is the least significant bit, which is bit 32 in the usual FITS convention and bit 0 in the usual C programmers convention.

5.6 Table filtering on vector columns using region filters

In CIAO, a 'vector column' is a paired column consisting of two components. For instance, the DET vector column has components DETX and DETY. You can see which of the columns in the file are vector columns by using 'dmlist filename cols'. There are two ways to filter on a vector column: define a rectangular region by filtering on each of the components as usual,

```
dmcopy "evt.fits[detx=4000:5000,dety=3000:4000]" rectangle.fits
```

or, use a region filter (see below for the full region syntax) on the vector column, either using its name - in this case DET - or the two components in parentheses - in this case (DETX,DETY).

```
dmcopy "evt.fits[det=circle(4500,3500,120)]" circle.fits
dmcopy "evt.fits[(detx,dety)=circle(4500,3500,120)]" circle.fits
```

5.7 Compound filters

The syntax supports compound (logical OR) filters:

```
dmcopy "evt.fits[(ccd_id=2,chipx=512:513)||(ccd_id=7,chipx=500:520)]"  two_bits.fits
```

Note that we do not support arbitrarily complex C-style logical expressions, just lists of filters separated by ||. In particular, nested parentheses are not supported.

5.8 Exclude filters

A very useful feature is to invert a filter, excluding instead of including:

```
dmcopy "evt.fits[exclude sky=region(reg.ds9)]" holes.fits
dmcopy "evt.fits[exclude pha=2:100,grade=7]" clean.fits
```

This is particularly useful in conjunction with compound filters:

```
dmcopy "evt.fits[exclude (ccd_id=0:6,8:9,chipx=513,fltgrade=208)||
                 (ccd_id=7,chipx=512:513,fltgrade=2,64)]" better.fits
```

It's very easy to make an exclude filter which results in an extremely complicated and unhelpful data subspace (where the positive filter rather than the exclusion must be stored). If you run into this problem, you can suppress the propagation of the subspace with [opt update=no].

```
dmcopy "evt.fits[exclude ccd_id=0:6,8:9][opt update=no]" test.fits
```

5.9 Filter syntax specification

A filter command is made up of filter components, which are made up of filter specifications. Filter specifications can be either a range filter, a relational filter or a region filter. We'll describe the smallest items first (filter specs) and work up to the complete filter command.

5.9.1 Filter variable names

In filters, we use the names of 'descriptors' in the file. The choices are:

- The name of a table column, image axis, or column/axis component. Examples:

  ```
  pha
  sky
  time
  ```

- A name beginning with the character # followed by digits, referring to a numbered column or axis. Examples:

  ```
  #1
  #5
  ```

- A pair of names separated by commas and enclosed in round parentheses, thus for example (detx,dety). This defines a 2-dimensional descriptor from two one-dimensional descriptors or component names.
- The special name #row is allowed for a filter command in a table block only. It points to a fake data descriptor whose value is the number of the row in the current input block, and so supports the ability to filter on row number.

5.9.2 Range filters

The range filter

```
pha = 1:10,80:300,51:100
```

is read as: 'the list of restrictions on pha is ...'.

- A range filter defines a restriction on a scalar data descriptor (a column in the table or a scalar axis in the image). It consists of the name of the descriptor followed by an equals sign, followed by a range list.
- If the name alone is used, without an equals sign or range list, it must be a quantity of logical data type.
- A range list is a comma-separated list of ranges.
- Each range defines a minimum and maximum value. In the most general form, the minimum and maximum are given separated by a colon. The range is a closed interval on the real line. Several abbrevated forms are supported:
  - **min:max**
    Range from min to max.
  - **min:**
    Range from min to the maximum legal value (TLMAX) of the quantity.
  - **:max** Range from the minimum legal value (TLMIN) to max.
  - **val**
    A single value is equivalent to val:val, a range with only one value.

Examples:

```
dmcopy "rh1012.fits[events][pha=5:10,20:100]" filtered.fits
```

```
dmcopy "rh1012.fits[events][grade=0,2:4,6,status=1:]" filtered.fits
```

```
dmcopy "rh1012.fits[events][x=:5.0,10.0:]" filtered.fits
```

5.9.3 Image sections

A special variant of range filters is the image section syntax, when the names of the quantities are omitted and only one range is permitted for each quantity. In this case, the quantities are assumed to be #1, #2, etc. in order. Thus for an image

```
ih1012.fits[4:8,1012:1234,10:20]
```

is equivalent to

```
ih1012.fits[#1=4:8,#2=1012:1234,#3=10:20]
```

If image section syntax is used, it may not be mixed with other kinds of filter command.

5.9.4 Relational filters

An alternative filter syntax

```
pha op value
```

may be read as 'true if pha op value'.

The supported operators are

- >, greater than
- <, less than,
- >=, greater than or equal
- <=, less than or equal
- ! =, not equal to **warning: may need to escape on unix command line**

Examples:

```
dmcopy "rh1012.fits[events][pha=5:10,x>100]" filtered.fits
```

```
dmcopy "rh1012.fits[events][grade=0,2:4,6,status\!=0]" filtered.fits
```

Unfortunately most unix shells require you to escape the !. When using parameter prompting, you won't need the backslash.

```
dmcopy "rh1012.fits[events][pha\!=5,x<2000]" filtered.fits
```

Note that you can't mix notations for a single variable, so

```
dmcopy "rh1012.fits[events][x\!=5.0:10.0]" filtered.fits
```

isn't allowed, to exclude a range you would have to do

```
dmcopy "rh1012.fits[events][x=:5.0,10.0:]" filtered.fits
```

instead.

You also can't do a general relational statement with nested parentheses and logical operators; the relational syntax is provided for familiarity, but in general the colon-syntax for specifying ranges is recommended.

5.9.5 Filter commands

- A filter specification may be either a range filter, a relational filter, or a region filter.

  ```
  pha=4:10,40:80
  time>400.0
  (detx,dety)=circle(400,200,10)
  ```

- A filter component consists of a comma-separated list of filter specifications; the complete filter component is the logical AND of all of the filter specifications.

  ```
  pha=4:10,40:80,time>400.0,(detx,dety)=circle(400,200,10)
  ```

- A standard filter command consists of only a single filter component.
- A filter command is an external filter command (see below), a standard filter command, or a compound filter command.

5.9.6 External filter commands

- An external filter command consists of the character @ followed by the name of a block (precisely, a virtual block command with only the dataset and block ID's), for example

  ```
  rh1012.fits[events][@tmp12.fits[2]]
  ```

  The data subspace of the specified block is opened and used to filter the input block.

- An external filter command may also be an ASCII file containing a valid filter specification. For example,

```
rh1012.fits[events][@my.filt]
```

where the file my.filt consists of the single line

```
pha=4:10,detpos=circle(54,52,2)
```

6 Regions and region filtering

```
[filter name=shape(...)]
[filter name=shape1(...)*shape2(...)+shape3(...)]
[filter name=field()-shape(...)]
[filter name=region(file)]
```

Regions are two dimensional filters that can be used to include or exclude data from a given file. They may be applied to any 'vector column' in the file.

Regions are made up of a number of shapes that are described below. The shapes can either be inclusive or exclusive. The different shapes can then be combined using either a boolean AND or boolean OR operation which leads to a degree of flexibility to design arbitrary regions.

The following region shapes are defined (see below for detailed explanation)

```
BOX(xcenter,ycenter,xlen,ylen)
ROTBOX(xcenter,ycenter,xlen,ylen,angle)
CIRCLE(xcenter,ycenter,radius)
RECTANGLE(xmin,ymin,xmax,ymax)
ANNULUS(xcenter,ycenter,iradius,oradius)
ELLIPSE(xcenter,ycenter,xradius,yradius,angle)
POINT(xcenter,ycenter)
SECTOR(xcenter,ycenter,minangle,maxangle)
PIE(xcenter,ycenter,iradius,oradius,minangle,maxangle)
POLYGON(x1,y1,x2,y2,x3,y3,...)
FIELD()
```

Shape arithmetic (boolean operation) is supported with the * (or &) operator which gives the intersection of two regions, the + (or |) operator which gives the union of two regions, and the - (or !) operator which gives the inverse of a region.

The FIELD() shape is given to allow you to exclude regions using

```
dmcopy evt[sky=field()-circle(4096,4096,20)] out.fits
```

which is equivalent to

```
dmcopy evt[exclude sky=circle(4096,4096,20)] out.fits
```

For the polygon shape, if the last point is not equal to the first point, another point will be added onto the end to close the polygon.

The syntax also supports the "region(file)" directive, which can read in ASCII region files produced by DS9, SAOImage, SAOtng, and PROS. Note that ds9 can write out region files in any of these formats. It can also read ASC-FITS REGION files. Also note that the region(file) directive cannot be mixed with other shapes.

6.1 Issues with CIAO regions

There are several subtleties with CIAO regions that are worth noting. These arise particularly when using region files made by imagers like DS9.

6.1.1 Celestial coordinate regions

In order for a CIAO tool to make use of a region in world coordinates, e.g.

```
  circle(06:37:11,-40:20:00,5')
```

it must know about the mapping between world and physical coordinates for the dataset in question. Not all CIAO tools have this ability yet. If you have problems with celestial coordinates, try using a region in physical pixel coordinates instead.

DS9 region files saved in celestial coordinates in degrees are not currently supported in CIAO; celestial coordinates are only recognized in sexagesimal format.

6.1.2 Stacks of single-shape regions versus regions with many shapes

It's important to understand the difference between two concepts: the REGION STACK and the REGION FILE.

A region file defines a single 'region' which may have multiple shapes. For example, a region file with

```
 circle(4096,4096,20)
+circle(5000,2000,100)
```

is a single region consisting of two circles. Applying a region file to some data lets CIAO decide whether points are in the region, but no knowledge is returned about which of the shapes a point is in.

A region stack contains multiple regions. CIAO will figure out each of the regions separately. The most common use of region stacks is in dmextract, where binning on a region stack creates a histogram where each bin corresponds to one region.

The syntax is

```
dmextract "evt.fits[bin sky=@reg.stk]" ...
```

to operate on a region stack and make a histogram of counts versus region. In contrast,

```
dmextract "evt.fits[bin sky=region(reg.stk)]" ...
```

specifies a single (albeit complicated) region, which will generate a single output bin.

6.1.3 Includes and Excludes

The CIAO syntax requires excluded regions to follow included regions. If you draw four circles on DS9, two includes and two excludes, and then save the region, the resulting file depends on the order the regions were drawn. If the order is -circle+circle-circle+circle, CIAO will not understand because of the leading -circle.

Note that

```
+circle(4096,4096,1000)
-circle(4096,4096,250)
+circle(4096,4096,50)
```

will have the inner 50-radius circle included, and the 50-250 radius annulus excluded, but

```
+circle(4096,4096,50)
-circle(4096,4096,250)
+circle(4096,4096,1000)
```

will include everything, as it's interpreted as "(R50 minus R250) OR R1000", and the R1000 includes everything. Therefore, you must be careful about order when generating region files in DS9.

6.1.4 Elliptical Annuli

CIAO doesn't understand the format DS9 uses for writing elliptical annuli, or for writing grids of annuli.

6.1.5 Region Areas (BACKSCAL)

For simple regions, the area can be calculated using analytical methods. For complicated regions where shapes overlap, or are close enough that it's not obvious they don't overlap, we must use a different method; CIAO divides up the area into bins (which may be smaller than a single pixel) and counts the number of bins which pass the test 'am I inside the region?'. This approach using discrete bins has a finite error, which is usually of order one percent of the area. CIAO2.2 is a bit smarter than CIAO2.1 and manages to do a better calculation in many cases, but in other cases the effect of the approximation may still be noticeable.

Region areas are most commonly used in the BACKSCAL keyword added to PHA/PI spectral files generated by dmextract. The error in the area is almost always small compared to other calibration uncertainties.

Be particularly careful when making regions comparable in size to the pixel size. When applying such a region to a table which contains fractional pixel positions, filtering does occur in exactly the area specified, although the user should be careful about interpretation - for instance, the true radial profile of a source may be blurred by aspect, PSF and instrument pixelization. When applying the region to an image file, filtering will accept all the counts in a pixel if the pixel center is within the region, and none of them if the pixel center is outside the region; therefore, the effective area sampled is not quite the same as the area calculated.

6.2 Examples

```
my_tool parameter="file.ext[EXTENSION][sky=circle(10,10,20)]"
```

Filter the file, a table, on the X and Y columns that includes only the data inside the circle described above. For a table you must specify which two columns to use (or which vector column).

Note that "sky" is the name of a vector column in the file and translates to "(x,y)".

Note also that the default behavior is to exclude the entire field and then start adding to it.

```
my_tool "foo[field() * !circle(30,40,10)]"
```

Filter the file "foo", an image, by including the entire image and then excluding a circular region (e.g. remove a source from the field). For an image with this syntax, the region is assumed to be in logical coordinates. The syntax is short hand for the more formal "foo[(#1,#2)=field()*!circle(30,40,10)]" To specify physical coordinates instead, use "foo[(x,y)=circle(30,40,10)]".

```
my_tool "foo[(energy,time)=region(ds9.reg)]"
```

Filter the file on the energy and time columns with the region stored in the file ds9.reg.

6.3 Syntax definition

- A region filter consists of a name followed by an equals sign followed by a region expression. The name must refer to a two-dimensional quantity.
- A region expression is a set of region elements joined with logical region operators: '&' or '*' performs a logical AND. '|' or '+' performs a logical OR.

  An element can be proceeded by an '!' or '-' to indicate that the area contained in the shape is to be excluded rather than included. However the first element in a region filter must not begin with '-'.

  ```
  dmcopy "acis_evts.fits[sky=field()-circle(4096,4096,100)]" background_evt.fits
  ```

  will copy all events from the field() region but will exclude events from the circle() region.

- The region elements define shapes in a two-dimensional plane. Each region element has two forms: the element name followed by round parentheses containing a comma-separated list of parameters, or the element name followed by a space-separated list of parameters terminated by a comma, a region operator, or the closing square parenthesis of the qualifier. Each region element name also has an abbreviated form. Coordinates can be given in pixels or hh:mm:ss.s, dd:mm:ss.s. Radii are in pixels or arcminutes (with $'$, e.g. $7'$). Angles are in degrees. Example: sky=circle(14:09:27,-00:02:33,5.3$'$).

- Allowed region elements are

field()	the whole image
circle(xcen,ycen,radius)	Center (xcen,ycen) with given radius
ellipse(xcen, ycen, rad1, rad2)	rad1 and rad2 are the semi-major and -minor axes
annulus(xcen, ycen, rad1, rad2)	rad1 and rad2 are the inner and outer radii
box(xcen, ycen, xlen, ylen)	xlen and ylen are the width and height of the box
rotbox(xcen, ycen, xlen, ylen, angle)	Same as box, but rotated through given angle in degrees
rectangle(xmin, ymin, xmax, ymax)	Like box, but specify lower left and upper right corners
rotrectangle(xmin, ymin, xmax, ymax, angle)	Like rectangle, but rotated
point(xcen, ycen)	A single point
pie(xcen, ycen, rad1, rad2, angle1, angle2)	An annular sector, with inner and outer radii and angles measured counterclockwise from +X
sector(xcen, ycen, angle1, angle2)	Like pie with rad1 = 0 and rad2 = infinity
polygon(x1,y1, x2,y2,....,xn,yn)	List of points to make closed polygon
region(filename)	ASCII region file

Examples:

```
dmcopy "acis_evts.fits[sky=region(saoimage.reg)]" out.fits
```

will read an SAOImage/DS9 or ASC-FITS region file with name "saoimage.reg".

7 Image Filtering

```
[filter nameA=min1:max1,nameB=min2:max2]
[filter #1=min1:max1,#2=min2:max2]
[min1:max1,min2:max2]
[filter nameA=shape(parameters),nameB=REGION(region_filename)]
[filter name=REGION(region_filename)][opt full]
[filter @filter.lis]
[shape(parameters)]
```

In the DM, an 'image' is a single N-dimensional rectangular array of data together with its metadata (header info, coordinate systems, etc). The concept maps closely to a single FITS image file, although in future releases we expect to be able to treat the contents of table array columns in the same way.

Although an image may have 1, 2, 3, or more dimensions, we provide special support for the case of a 2-dimensional image, and the examples below assume such an image.

7.1 Image coordinates

One complication with images is the distinction between the actual pixels of the image (which we call 'logical' coordinates, and the DS9 display program calls 'image' coordinates) and the original pixels (which we call 'physical coordinates') of the data from which the image was created. Sometimes these are the same, but often we make images at reduced resolution. For example,

```
dmcopy "evt2.fits[bin detx=3000.5:5000.5:2.0,dety=4000.5:6000.5:2.0]"  by2.img
```

makes an image which is 1000 x 1000 pixels in size. Its logical pixel numbers then run from 1 to 1000, and so the logical coordinate values run from 0.5 (left hand edge of first pixel) to 1000.5 (right hand edge of last pixel).

To distinguish beween the logical and physical systems, we use the notation "#1,#2,.." to denote the 1st, 2nd.,... logical coordinate axes, and actual variable names "detx,dety,.." etc. to denote the corresponding physical axes. Non-CIAO FITS images without axis names are given the default physical axis names X,Y,Z (if N is up to 3). To see the physical axis names on your image, do

```
dmlist by2.img cols
```

In this example, logical pixel #1=0.5 corresponds to physical pixel x=3000.5.

7.2 Image filtering on logical coordinates

To filter an image on its logical coordinates,

```
  dmcopy "im.fits[#1=257:512,#2=1:100]" subset.fits
```

pulls out a 256 x 100 subset of the image.

```
  dmcopy "im.fits[(#1,#2)=circle(145,356,25)]" circle.fits
```

sets everything outside the specified circle to zero. The default behaviour of a filter like this is to also shrink the resulting image to be as small as possible surrounding the circle. To suppress this behaviour, and get a big, almost empty, image (the same size as im.fits) with a small circle in the middle, use the "opt full" modifier as shown below:

```
dmcopy "im.fits[(#1,#2)=circle(145,356,25)][opt full]" circle.fits
```

7.3 Image filtering on physical coordinates

To filter on the original physical coordinates, we refer to the physical axis names:

```
dmcopy "im.fits[x=4096.5:4213.5,y=4142.3:6120.1]" subset.fits
```

You can also use a region filter:

```
dmcopy "im.fits[(x,y)=circle(4096,4096,12)]" circle.img
```

This will give you a square image bounding the given circle, and set to zero all pixels outside the circle. If you wish to mark these pixels as "invalid", you can use the "opt null" syntax described in "ahelp dmopt":

```
dmcopy "im.fits[(x,y)=circle(4096,4096,12)][opt null=-999]" circ2.img
```

Note that not all programs will recognise that such pixels are to be ignored, although both ds9 and dmstat do.

7.4 Image filtering on world coordinates

You can also use world coordinates in a region filter:

```
dmcopy 'im.fits[(x,y)=circle(11:03:28.4,-20:11:23.2,20.1\")]' circ.img
```

However, you can't use the world coordinates when filtering on a one dimensional range - i.e. [x=11:03:28.2:11:03:32.1] won't work, you would have to do use "(x,y)=rectangle(...)" instead (and of course, x and RA are not quite parallel in the tangent plane projection).

Using (x,y) is a bit of a cheat in the syntax; we should really use (RA,Dec) to signal to the DM that world coordinates are being used, which would let us also support entering those world coordinates in decimal degrees. That will probably be added in a later release.

8 Column selection

8.1 Description

The select command selects columns for an output table. For an image, it specifies the names and ordering of output axes, and is interpreted as a bin command. A bin command and a select command may not be present in the same virtual block specification.

```
[cols name1,name2,...]
[cols #1,#4,...]
[cols -name1,-name2]
[cols newname1=name1,newname2=name2,...]
```

The DM virtual file syntax lets you pick out selected columns from a table.

```
dmcopy "evt.fits[cols time,pha]" tmp1.fits
```

makes a file containing only the columns time and pha.

```
dmstat "evt.fits[cols grade]"
```

passes to dmstat a virtual file containing only the column called "grade".

```
dmcopy "pi.fits[cols rate=count_rate]" npi.fits
```

renames the column count_rate to be simply rate.

```
dmstat "evt.fits[cols #3]"
```

passes to dmstat a virtual file containing only the third column in evt.fits (where 'column' is used in the DM sense, in which sky(x,y) is a single "vector" column; use "dmlist evt.fits cols" to find the column numbers).

You can also exclude columns:

```
dmcopy "evt.fits[cols !status]" tmp1.fits
```

```
dmcopy "evt.fits[cols -status]" tmp1.fits
```

makes a file without the status column. The first form needs to be typed as "" in the Unix shell, so the second form may be more convenient.

```
dmlist "evt.fits[cols ra,dec]" data
```

```
or dmlist "evt.fits[cols eqpos]" data
```

lists the values of pseudo-columns RA and DEC, which are coordinates defined on the sky(x,y) column. You can find the names of the coordinate systems defined for a file by using the 'cols' option to dmlist:

```
dmlist evt.fits cols
```

and then using these coordinate system names in a DM [cols ...] qualifier lets you turn them into actual columns. Note that in release 2.2, the coordinate names work in the [cols ...] qualifier but not in the filtering and binning qualifiers. If you want to explicitly filter on a coordinate, you must first use dmcopy with [cols *,ra,dec] to explicitly create the new columns and then filter the resulting file.

8.2 Syntax definition

- The select command is recognized by the parser as a qualifier which begins with the word COLUMNS (or COLS) followed by a space followed by other text.
- The select text consists of a comma separated list of column specifications.

- The reserved column specifications #all (synonym *) and #none (synonym -) denote selection of all of the columns of the input table or none of them respectively.
- Omitting the select command is equivalent to [columns *].
- The column specification newname=oldname renames a column from oldname to newname.
- The column name #n, where n is an integer, denotes the nth column in the base block.
- A column specification is the name of a column or of a coordinate system (which is promoted to a column in the virtual file).
- The specification !name means delete that column. It is only useful when no other columns are positively selected. Thus

  ```
  rh1012.fits[events][columns !pha,!status]
  rh1012.fits[events][columns *,!pha,!status]
  ```

  each make a table with all the columns of [events] except pha and status, but

  ```
  rh1012.fits[events][columns time,!pha,!status]
  ```

  makes a table with only time, and the !pha,!status commands are redundant.

9 Binning

9.1 Description

Summary:

```
[bin nameA=min:max:step]
[bin nameA=min:max:#bins]
[bin nameA=min:max:step,nameB=min:max:#bins]
[bin/f nameA=min:max:step,nameB=min:max:step]
[bin vname=step]
```

By ‘binning’ we mean making n-dimensional histograms of a quantity. A typical binning operation is to define a 2-dimensional image by taking two columns of values, say x and y, from a table and histogramming them; each pixel in the image corresponds to a range in x and y, and the pixel value is equal to the number of entries in the table whose x and y lie in that range. The binning directive can also be used to rebin an existing image to a new pixel size.

For example: Consider a small table with three columns A, B, C and five rows:

```
Row A B C
1 1.0 2.1 1.8
2 2.8 1.2 4.2
3 2.5 1.3 4.1
4 3.6 2.0 0.3
5 2.0 2.2 5.1
```

We can "bin on A and B" by making a two-dimensional histogram, thinking of A and B as spatial coordinates. The DM directive

```
[bin A=0.5:3.5:1,B=0.5:3.5:1]
```

is read "bin A from 0.5 to 3.5 in steps of 1, and B from 0.5 to 3.5 in steps of 1". This generates a 3 x 3 grid in which the lower left bin runs in A from 0.5 $<= A < 1.5$, and so has center (1.0, 1.0). If we take the table above we obtain the array (as it would be seen in an image display tool)

```
A=1 2 3
--------------
B= 3 | 0 0 0
B= 2 | 1 1 0
B= 1 | 0 0 2
```

We can make a coarser or finer grid by altering the step size since row 1 falls in bin (1,2), row 2 falls in bin (3,1), and so does row 3 (because the A=2.5 is rounded up), and row 5 falls in (2,2). Row 4 is outside the grid and so is discarded; and because we are binning on A and B, the information in the C column is discarded. This process is how we turn event lists into images,

```
dmcopy "evt.fits[bin x=0.5:8192.5:4,y=0.5:8192.5:4]" img.fits
```

or (in the 1-dimensional case) light curves or spectra.

```
dmcopy "evt.fits[bin time=74321940.2:74328100.0:10.0]" lc.fits
dmcopy "evt.fits[bin energy=1000:7000:10]" energy.fits
dmextract "evt.fits[bin pi=1:1024]" standard.pha.fits
```

9.2 Binning tables into images

- A binning directive starts with the word 'bin' or 'bin/f', followed by a space, followed by the binning list.
- The binning list is a comma-separated list of binning specifications, each of which defines one axis or axis group.
- The simplest kind of binning specification is a name. The name specifies that that axis is next in the ordering. If the base block is a table, the corresponding column will be binned on.
- The name may be followed by an equals sign and a binning. The binning specifies a start value, a stop value and a step size in the form min:max:step; alternatively, the number of bins may be given instead of the step size, in which case this number is preceded by a hash sign. The defaults are TLMIN, TLMAX and 1.0. Alternate forms supported are:
  - min:max:step
  - min:max
  - min:
  - :max
  - step
  - ::step
  - min::step
  - :max:step
  - min:max:#nbins

  The only potential parsing confusion is with the case with no colons, where there is conflict with a column/axis name whose name is the same as the step size. For instance, does

  ```
  rh1012.fits[bin pha=4]
  ```

  mean 'bin pha by 4' or 'bin a column called '4' by 1 and rename it pha'? Another good reason to not give columns numeric names. In this case we will require columns in the binning spec to not have numeric names - if you do have such a column, you'll have to use the #n notation to reference it.

For vector columns, you should specify the binning separately for each component if a min and max are required:

```
[bin x=0.5:8192.5:4,y=0.5:8192.5:4]
```

but if the full range is selected and only a step size is specified, the vector column name may be used:

```
[bin sky=4]
```

If no explicit binning is given, a default binning will be used. For an unfiltered table column, the axis will extend across the full valid range (FITS TLMINn to TLMAXn) in steps of size 1.0. Example:

```
evt.fits[events][bin pha,time]
```

makes a pha versus time image.

If the table column has been filtered, the resulting image will be shrunk to match the filter. In particular,

```
evt.fits[events][sky=circle(4096,4096,200)][bin sky=4]
```

will make an image just big enough to contain the radius 200 circle that has been filtered on (it will be 100 x 100 binned pixels in size). Of course, image arrays have to be rectangular (by the definition of a matrix) - the fact that pixels outside the circle are not valid data is encoded in the data subspace. They are set to zero by default, but this can be changed using the [opt null=value] directive.

Sometimes it is important to retain the original range of the variable without shrinking the output, even if that results in a lot of blank space in the image. The bin/f directive (for "bin/full") forces the full size to be used:

```
evt.fits[events][sky=circle(4096,4096,200)][bin/f sky=4]
```

9.3 Columns with integer data types

For columns of integer data type, the directive

```
[bin x=1:1024:2]
```

is taken to be inclusive, that is $1 <= x <= 1024$, so that the first bin consists of x=1 and x=2.

You can omit any or all of the three values (min,max,step) in the binning specification. The default value of the step size is almost always 1.0 (it is possible to set another default in the data file by adding a TDBINn keyword to the header), so [bin x=1:1024] is the same as [bin x=1:1024:1]. The default values of the min and max are found from the descriptor ranges in the data file (for FITS files, these are the TLMINn/TLMAXn keywords). These default values are displayed when you type 'dmlist filename cols'. If you have a column for which the descriptor ranges are not set, the DM can't guess a default and you must use an explicit range.

If the defaults are present in the file, then [bin x] will select the default values; [bin x=::2] will select the default range with a step size of 2; [bin x=2] is accepted as a synonym for this. To use one default, [bin x=:512] or [bin x=12:] or [bin x=12::1] are all acceptable.

Another layer of defaulting is provided by the mission database file cxo.mdb, which currently defines standard binning for PHA spectral extractions. In the current release, the only program that recognizes this file is dmextract (see the defaults parameter of dmextract).

Note that you can use a real-valued binning factor, which may be less than one: [bin x=::0.5] makes a double resolution image.

If you specify a bin size which doesn't divide evenly into the binning range, the last bin will be 'incomplete'. For instance, [bin x=1:10:3] creates four bins (1:3),(4:6),(7:9),(10:12), but the final (10:12) bin will only include contributions from rows with x=10.

An alternate syntax to specify the total number of bins instead of the bin size is [bin x=1:1000:#20], which says "bin from 1 to 1000 making a total of 20 equal size bins".

9.4 Columns with real data types

For columns of real data type, be careful when binning coordinate values. The ACIS sky coordinate space runs from 0.5 to 8192.5, for a total of 8192 pixels; pixel (512,512) runs from 511.5 to 512.5 in each axis. Binning [x=1:3:1] makes only 2 bins: (1.0:2.0) and (2.0:3.0); contrast this with the integer-column case where three bins are made: (0.5:1.5), (1.5:2.5), and (2.5:3.5).

Otherwise, the syntax is the same as for integer columns.

9.5 Binning with range and region filters

Frequently you may wish to combine filtering and binning. For example,

```
dmcopy "evt2.fits[ccd_id=7][bin chip]" chip7.img
```

makes an image in chip coordinates, first filtering the event list to accept only events on ccd 7.

Usually you won't filter on the same columns that you're binning on; it's sufficient to set the range:

```
dmcopy "srclist.fits[flux=100:200][bin flux]" flux_hist.fits
```

while perfectly valid, is better expressed as

```
dmcopy "srclist.fits[bin flux=100:200]" flux_hist.fits
```

In the CIAO1 release, there was a significant difference: the first of these forms would give you an output image over the full default range of the variable 'flux', instead of just the range of interest 100:200, with zeroes in the image outside that range of interest. Sometimes you may actually want this, when you are planning to sum or match different images. As discussed above, the special syntax bin/full or bin/f reproduces the CIAO1 behaviour:

```
dmcopy "srclist.fits[flux=100:200][bin/f flux]" flux_hist.fits
```

This is particularly relevant when applying sky region filters:

```
dmcopy "evt2.fits[sky=circle(4096,4096,100)][bin sky]" circle1.fits
```

makes a 201 x 201 pixel image with all the pixels outside the circle set to the null value (default zero). However,

```
dmcopy "evt2.fits[sky=circle(4096,4096,100)][bin/f sky]" circle1.fits
```

makes an 8192 x 8192 pixel image (the default range of SKY) with all the pixels outside the circle zero (this was the standard CIAO1 behaviour).

10 Name-command

The name-command allows the user to specify the name of the virtual block. *(This only works for a simple block renaming at the moment)*:)

- For a virtual table specification, the name command must be a simple name, which names the virtual table block.
- If the name command is absent, the virtual block has the same name as the input block.
- If the name command is present, another qualifier must also be present to avoid ambiguity with the block identifier. Thus

  ```
  rh1012.fits[events]
  ```

  specifies an input block name, while

  ```
  rh1012.fits[][events]
  ```

  specifies a virtual block name.

11 Fine-tuning using the opt directive

11.1 Description

The [opt ...] qualifier in the DM virtual file syntax lets you control special details of the virtual file.

```
[opt name=value,...]
```

For example,

```
dmcopy "evt.fits[sky=circle(4096,4096,200)][bin sky][opt null=-99.0,type=r4]" new.img
```

makes an image file of data type 4-byte real in which the pixels outside the filter circle are set to a value of -99.0.

11.2 Supported options

11.2.1 type: Image Data Type

```
[opt type={value}]
```

This option, used when binning a table to an image, forces the output data type. Supported values are i2,i4,r4,r8 for 2 and 4 byte integers and 4 and 8 byte reals. The default value is i2.

11.2.2 null: Image Null Value

```
[opt null={value}]
```

An image array is by definition rectangular, but we often want to make an image including only data from, e.g., a circular region. The pixels in the output image outside the circle are not valid. When copying an image through a filter, or binning a table to an image, this option controls the value of pixels which lie outside the filtered area. The default behaviour is to set these pixels to zero; [opt null=...] sets them to a specified value, which of course must be of the same data type as the output image.

The special value "NaN" is supported, but is only valid if the output image is of real data type:

```
dmcopy "evt.fits[sky=circle(4096,4096,200)][bin sky]
 [opt null=NaN,type=r4]" nan.img
```

11.2.3 mem: Maximum memory to use

```
[opt mem={value}]
```

Set the maximum memory used for a virtual image, in megabytes. The DM has an internal default limit on the size of a binning operation; this prevents users from accidentally commanding the DM to create a many-gigabyte image on a machine which only has a hundred megs of memory. If you really want to create a big image, you use this option to override the internal default. Example:

```
dmcopy "evt.fits[bin sky=4][opt mem=100]" big.img
```

11.2.4 full: Full image size

```
[opt full]
```

Binning a filtered table to an image will by default select the smallest image size that can contain the filtered data. Thus

```
dmcopy "evt.fits[sky=circle(4000,4000,100)][bin sky]" circ.img
```

is equivalent to

```
dmcopy "evt.fits[sky=circle(4000,4000,100)][bin x=3900:4100,y=3900:4100]" circ.img
```

rather than the full field image

```
dmcopy "evt.fits[sky=circle(4000,4000,100)][bin x=0.5:8192.5,y=0.5:8192.5]"
  circbig.img
```

If you explicitly want to retain the full field even though most of it will be empty (or with values set by the [opt null=value] command) you can use either the bin/f command, or its equivalent the 'opt full' directive:

```
dmcopy "evt.fits[sky=circle(4000,4000,100)][bin sky][opt full]"
     circbig.img
```

11.2.5 update: Suppress subspace propagation

```
[opt update=no]
```

For complicated exclude filters, the process of updating the data subspace may be long and not very helpful. The update=no option suppresses recording of the current filter:

```
dmcopy "evt.fits[sky=region(nasty.reg)][opt update=no]" nasty.fits
```

11.2.6 scale: Suppress FITS autoscaling

```
[opt scale=no]
```

The default behaviour in the current DM release is to automatically apply internal (BSCALE/TSCAL) scaling for FITS files. This option turns off the scaling, allowing access to the raw values.

12 Getting help

If you can't find what you need in this document, try the threads page at asc.harvard.edu/ciao, and if that fails send email to ascds_help@cfa.harvard.edu.